Group-Based Parallel Multi-scheduling Methods for Grid Computing

Abraham, Goodhead Tomvie

Award date:
2016

Awarding institution:
Coventry University

Group-Based Parallel Multi-scheduling Methods for Grid Computing

Goodhead Tomvie Abraham

June 2016

A thesis submitted in partial fulfilment of the University's requirements for the degree of

Doctor of Philosophy

Faculty of

Engineering and Computing

Coventry University

Abstract

With the advent in multicore computers, the scheduling of Grid jobs can be made more effective if scaled to fully utilize the underlying hardware and parallelized to benefit from the exploitation of multicores. The fact that sequential algorithms do not scale with multicore systems nor benefit from parallelism remains a major challenge to scheduling in the Grid. As multicore systems become ever more pervasive in our computing lives, over reliance on such systems for passive parallelism does not offer the best option in harnessing the benefits of their multiprocessors for Grid scheduling. An explicit means of exploiting parallelism for Grid scheduling is required. The Group-based Parallel Multi-scheduler for Grid introduced in this work is aimed at effectively exploiting the benefits of multicore systems for Grid job scheduling by splitting jobs and machines into paired groups and independently multi-scheduling jobs in parallel from the groups. The Priority method splits jobs into four priority groups based on job attributes and uses two methods (SimTog and EvenDist) methods to group machines. Then the scheduling is carried out using the MinMin algorithm within the discrete group pairs. The Priority method was implemented and compared with the MinMin scheduling algorithm without grouping (named ordinary MinMin in this research). The analysis of results compared against the ordinary MinMin shows substantial improvement in speedup and gains in scheduling efficiency. In addition, the Execution Time Balanced (ETB) and Execution Time Sorted then Balanced (ETSB) methods were also implemented to group jobs in order to improve on some deficiencies found with the Priority method. The two methods used the same machine grouping methods as used with the Priority method, but were able to vary the number of groups and equally exploited different means of grouping jobs to ensure equitability of jobs in groups. The MinMin Grid scheduling algorithm was then executed independently within the discrete group pairs. Results and analysis shows that the ETB and ETSB methods gain still further improvement over MinMin compared to the Priority method. The conclusion is reached that grouping jobs and machines before scheduling improves the scheduling efficiency significantly.

Acknowledgement

My deepest and sincere appreciation goes to my Director of Studies Professor Anne James whose tremendous support, commitments, suggestions, ideas, encouragement and mentorship helped shaped this work and moulded me. She contributed immensely to bring this work this far. I will forever remain grateful as I keep thanking my stars for having you with so much wealth of knowledge and experience as my Director of Studies.

My special thanks and appreciation also go to my project supervisor Dr.NorlailyYaacob who inspired me and this work in many ways. I am grateful for all your support, suggestions, brilliant criticisms and corrections and for keeping me focused on the work..

I also wish to thank and appreciate my third supervisor Dr. Saad Amin and Dr. Reda Albodour whose early contributions helped point out a direction for the research. I also wish to extend my gratitude to Prof. Kuo-Ming Chao whose contributions and observations especially during the PRPs contributed greatly to shape this work.

I owe a depth of gratitude to all the staff and research colleagues in Coventry University, words cannot express my gratitude for the role you all played at different stages of this work, I am deeply grateful.

My gratitude also goes to the management of the National Information Technology Development Agency (NITDA) (who in their quest to bridge the information technology gap between Nigeria and the Western world and place Nigeria on the IT map) granted me scholarship and availed me the opportunity to undertake this research. My most profound appreciation goes to the Niger Delta University management and the government of Bayelsa State of Nigeria for approving my study leave to enable me embark on this dream journey.

Most importantly, I would extend my special thanks and appreciation to my family for all the sacrifices they made on my behalf during this period. To my wife Mrs Goodhead Enimokie, thank you for all your prayers which strengthened me. Also, thank you for playing both the role of a father and mother for our children while I was away.

To my very young children: Rodney, Jahsmine, Wyse, Richarmah, Anne-Okievei and King-Giasue. I owe you all so much for the vacuum my absence created in your young and formative lives. I know how much you all miss me.

I am also grateful to my brothers: Millionaire, Marshal, and Benjamin and sisters Census, Edith and Izibelogo for their persistent prayers and unflinching support at every point of my trying time. To my friends, I say a big thank you for your various support and encouragement during this period.

I owe a dept of gratitude to the following persons: HRH King Godwin Gurosi Igodo (The Obene-Ibe of Atissa Kingdom), Dr. and Mrs. Godwin T. Igodo, Bar. Esueme Dan Kikile, Mr. Enime Godwin Yakiah, Mr. Kemi Ungbuku, Dr. Augustin Timbiri and Dr. Ovienadu Torutein.

And finally, to all the members of Bayelsa Focus Group (BFG), I say a big thank you for remaining behind the scene to provide me with virtual company at those lonely periods during this sojourn.

Dedication

To the memory of my mother Mrs. Balafakuma Abraham (nee Reuben) who despite all odds ensured that academic pursuit remains one and only option for me, dedicated her life towards achieving that cause but died at the brinks of the goal becoming a reality.

Published Articles

Journal Contributions

1. Abraham, G. T., James, A., and Yaacob, N. (2015a) 'Priority-Grouping Method for Parallel Multi-Scheduling in Grid'. Journal of Computer and System Sciences, (81)6, 943-57
   DOI: http://dx.doi.org/10.1016/j.jcss.2014.12.009

2. Abraham, G. T., James, A., and Yaacob, N. (2015b) 'Group-based Parallel Multi-scheduler for Grid Computing' Future Generation Computer Systems, 50, 140-153
   DOI: http://dx.doi.org/10.1016/j.future.2015.01.012

Conference Contributions

1. Abraham G. T., James, A., and Yaacob, N (2014) *Group-Based Parallel Multi-scheduler for Grid Computing* [Poster presentation] 'Coventry University and University of Warwick Branch BCS Event'. Warwick: Computer Science Department University of Warwick, 19th Feb 2014

2. Abraham G. T., James, A., and Yaacob, N (2014) *Group-Based Parallel Multi-scheduler for Grid Computing (revised)* [Poster presentation] 'Coventry University Annual Faculty Research Symposium'. Coventry: Faculty of Engineering and Computing, Coventry University, 26th Feb 2014

Table of Contents

Abstract i
Acknowledgement ii
Dedication iv
Published Articles v
Table of Contents vi
List of Equations xi
List of Tables xii
List of Figures xiii
Acronyms xv
CHAPTER ONE 2
INTRODUCTION 2
1.1 Introduction 2
1.2 Background to Problem 2
1.3 Motivation for undertaking this Work 3
1.4 Research Question 4
1.5 Aim and Objectives 5
1.6 Method 6
1.7 Contributions 7
1.8 Organization of Thesis 8
1.9 Summary 9
CHAPTER TWO 12
LITERATURE REVIEW 12
2.1 Introduction 12
2.2 The Grid 12
2.2.1 Overview 12
2.2.2 Architecture of the Grid 14
2.2.3 Main Types of Grid 15
2.3 Parallelism 17
2.3.1 Parallelism and Multicore Systems 17
2.3.2 Exploiting Parallelism in Multicore Systems 21
2.3.3 Multicore Systems and Constraints 22
2.3.4 Some Impediments to the Impact of Multicore Systems 28
2.4 The Grid and Parallelism 29
2.5 Distributed and High Throughput Computing (HTC) Systems 32

2.5.1 Examples of Distributed Systems 33
2.5.2 Parallel and distributed computing models/offerings 37
2.6 Parallel Scheduling Algorithms 41
2.6.1 Tree, Graph and Hypercube Parallel Scheduling Algorithms 41
2.6.2 Nature Inspired Algorithms 44
2.7 Grid Scheduling Algorithms 48
2.7.1 Classical Grid Scheduling Algorithms 50
2.7.2 Fusion and Enhancement of the Classical Algorithm 51
2.7.3 QoS-Focused Algorithms 52
2.7.4 Adaptive Grid Scheduling Algorithms 55
2.7.5 Nature Inspired Algorithms for Grid Scheduling 56
2.8 Parallelisation of the Grid Scheduling Task 58
2.8.1 Problems with Current Scheduling Algorithms 58
2.8.2 Parallelisation of the Grid Scheduling Algorithms 59
2.9 Group Scheduling and Load Balancing 61
2.9.1 Gang Scheduling 61
2.9.2 Grouping of Jobs 62
2.9.3 Relationship of this Research to Previous Research in Grouping 64
2.9.4 Load Balancing 66
2.10 Summary 67
CHAPTER THREE 70
RESEARCH QUESTION AND METHODOLOGY 70
3.1 Introduction 70
3.2 The Identified Gap 70
3.3 Overview of Method 70
3.3.1 Literature Review 71
3.3.2 Definition of Terms 71
3.3.3 Research Question Development 71
3.3.4 Solution Design and Development 71
3.3.5 Simulation 74
3.3.6 Experimentation 74
3.3.7 Analysis of Results 77
3.3.8 Evaluation of Results 81
3.3.9 Motivation for using MinMin for Comparison 82
3.4 Summary 84
CHAPTER FOUR 86
DESIGN OF THE GROUPING BASED MULTI-SCHEDULER 86
4.1 Introduction 86
4.2 Design of the Group-based Parallel Multi-Scheduler 86
4.2.1 Functions of the Group-based Parallel Multi-scheduler 87

4.2.2 The 'Shall Statement' and System Requirement 87
4.2.3 Context Diagram 88
4.2.4 Use Case Diagram 91
4.2.5 Activity Diagram 91
4.2.6 Sequence Diagram 94
4.2.7 Class Diagram 95
4.3 The GPMS 98
4.3.1 Overview of Processing 98
4.3.2 GPMS Job and Machine Grouping 100
4.4 Job Grouping Methods 102
4.4.1 Design of the Priority Method 102
4.4.2 Design of the Execution Time Balanced (ETB) method 106
4.4.3 Design of the Execution Time Sorted and Balanced (ETSB) method 107
4.4.4 Job Attributes and Job Categorization 109
4.5 Machine Grouping 109
4.5.1 Design of SimilarTogether (SimTog) Method 110
4.5.2 Design of EvenlyDistributed (EvenDist) Method 111
4.6 Experimental Testbed and Simulations 112
4.6.1 Grid Site 112
4.6.2 Grid Machines 113
4.6.3 Simulation of Grid, CPU Speed and Number of Cores 114
4.6.4 Local Policy 116
4.6.5 Source of Jobs to the System 117
4.6.6 Simulation of Priority and Execution Time 119
4.6.6 Executing Dynamically Generated Jobs 122
4.7 Experimental Design 123
4.7.1 The Experiments 124
4.7.2 Relationship between a job, a thread and a group 126
4.7.3 The Grouping of Jobs and Machines in GPMS 127
4.7.4 Combination of the Number of Experiments 127
4.8 Shortcomings of the Grid Workload Archive 128
4.9 Summary 129
CHAPTER FIVE 132
RESULTS AND ANALYSIS OF THE GPMS METHODS 132
5.1 Introduction 132
5.2 Results and Performance Evaluation of the Priority Method 132
5.2.1 Presentation of Results (Priority) 132
5.2.2 Discussion of Results (Priority) 140
5.3 Results, Analysis and Evaluation of the ETB Method 142
5.3.1 Presentation of Results (ETB) 142
5.3.2 Discussion of Results (ETB) 155

5.4 Results, Analysis and Evaluation of the ETSB Method 155
5.4.1 Presentation of Results (ETSB) 155
5.4.2 Discussion of Results (ETSB) 168
5.5 Comparative Analysis of the Group-based Scheduling Methods 169
5.5.1 Comparison between ETSB and ETB methods 169
5.5.2 Comparison between Priority, ETB and ETSB methods 172
5.5 Statistical Tests 174
5.6 Summary 188
CHAPTER SIX 192
GENERAL DISCUSSION ON RESULTS AND OUTCOMES 192
6.1 Introduction 192
6.2 Overview of Approach and Results 192
6.3 Priority Method 193
6.4 The ETB and ETSB Methods 194
6.5 Differences between ETB and ETSB Methods 196
6.6 Comparison of the ETB, ETSB and the Priority Methods 196
6.7 Comparison of Machine Grouping Methods (EvenDist and SimTog) 196
6.8 Load Balancing in the GPMS 197
6.9 Impact of shared resource contention on the overall result 197
6.9.1 Impact of thread contention between the GPMS and MinMin 197
6.9.2 Impact of thread contention between successive groups within the GPMS method 198
6.9.3 Impact of thread contention on makespan in the GPMS 199
6.10 Summary of Findings 199
6.11 Summary 200
CHAPTER SEVEN 202
COMPARISON OF GPMS AND PREVIOUS RESEARCH 202
7.1 Introduction 202
7.2 The Simulation Approach 202
7.3 Some Grid Simulation Tools 203
7.3.1 OptorSim 203
7.3.2 SimGrid 204
7.3.3 MicroGrid 204
7.3.4 GridSim 205
7.4 The GPMS Simulator 212
7.5 Comparison between GridSim and the GPMS simulator 218
7.5.1 Application Model 218
7.5.2 Resource Model 219
7.5.3 General Features 219
7.6 Relationship of the GPMS System to Gang Scheduling 220
7.6.1 Gang Scheduling 220

7.6.2 Gang Scheduling and the GPMS 222
7.7 Comparison between the GPMS and Condor 223
7.7.1 Condor 223
7.7.2 The heterogeneity of computers available to Condor 225
7.7.3 Gang Scheduling in Condor 226
7.7.4 GPMS and Condor Comparison 227
7.8 Relationship to DIANE 230
7.8.1 DIANE 230
7.8.2 Comparison between DIANE and the GPMS system 231
7.9 Summary 232
CHAPTER EIGHT 234
CONCLUSION AND FUTURE THOUGHTS 234
8.1 Introduction 234
8.2 Contributions to Knowledge 234
8.3 Conclusion 236
8.4 Future Thoughts 236
References 240
Glossary 264
Appendix A: Header File from the Grid Workloads Archive 268
Appendix B: Grid Workloads Archive Acknowledgement 271
Appendix C: Selected Job Scheduling Algorithms on the Grid 272
Appendx D: Some Research that employed the MinMin Scheduling Algorithm for Comparison 280
Appendix E: Project Ethical Approval 284

List of Equations

EQUATION 1 SPEEDUP (X) 78
EQUATION 2 SPEEDUP (%) 79
EQUATION 3 IMPROVEMENT OVER MINMIN (X) 79
EQUATION 4 IMPROVEMENT OVER MINMIN (%) 79
EQUATION 5 IMPROVEMENT BETWEEN GROUPS (X) 80
EQUATION 6 IMPROVEMENT BETWEEN GROUPS (%) 81

List of Tables

TABLE 1 BENEFITS OF PARALLEL COMPUTING 30
TABLE 2 SCHEDULING EXPERIMENTS 76
TABLE 3 NUMBER OF VARIATIONS OF EACH EXPERIMENT 76
TABLE 4 FUNCTIONS OF THE GPMS 87
TABLE 5 FUNCTIONS OF THE GPMS COMPONENTS 88
TABLE 6 ALGORITHM FOR THE GPMS 101
TABLE 7 ALGORITHM FOR THE PRIORITY METHOD 104
TABLE 8 SCHEDULING STEPS USING THE PRIORITY METHOD 105
TABLE 9 ALGORITHM FOR THE ETB METHOD OF GROUPING JOBS 107
TABLE 10 ALGORITHM FOR THE ETSB METHOD OF GROUPING JOBS 108
TABLE 11 ALGORITHM FOR THE SIMTOG METHOD OF GROUPING MACHINES 110
TABLE 12 ALGORITHM FOR THE EVENDIST METHOD OF GROUPING MACHINES 111
TABLE 13 FEATURES AND CHARACTERISTICS OF A GRID SITE 113
TABLE 14 FEATURES AND ATTRIBUTES OF A GRID MACHINE 114
TABLE 15 CHARACTERISTICS AND COMPONENTS OF THE SIMULATED GRID 115
TABLE 16 SELECTED ATTRIBUTES FROM THE GRID WORKLOADS ARCHIVE'S TRACE FILE 118
TABLE 17 EXAMPLE ROWS OF VALUES (RELEVANT ATTRIBUTES ONLY) FROM THE GWF TRACE FILE 119
TABLE 18 PSEUDO CODE FOR ESTIMATING SIZE OF JOBS 122
TABLE 19 ALGORITHM FOR SIMULATING EXECUTION TIME OF JOBS 122
TABLE 20 RESULTS AND COMPUTATION OF CORRELATION, ANOVA AND STANDARD DEVIATION (PRIORITY). 135
TABLE 21 PERFORMANCE IN MULTIPLES AND IN PERCENTAGE 136
TABLE 22 SPEEDUP IN PERCENTAGE AND IN MULTIPLES 136
TABLE 23 RESULT AND SPEEDUP FOR MINMIN AND ETB-EVENDIST 146
TABLE 24 RESULTS AND SPEEDUP FOR MINMIN AND ETB-SIMTOG 147
TABLE 25 ANOVA RESULTS FOR ETB-EVENDIST, MINMIN AND BETWEEN GROUP CARDINALITY 148
TABLE 26 PERFORMANCE OF ETB-EVENDIST AGAINST MINMIN AND BETWEEN GROUPS 148
TABLE 27 PERFORMANCE OF ETB-SIMTOG AGAINST MINMIN AND BETWEEN GROUPS 149
TABLE 28 SCHEDULING TIMES AND SPEEDUP FOR MINMIN VS. ETSB-SIMTOG 159
TABLE 29 SCHEDULING TIMES AND SPEEDUP FOR MINMIN VS. ETSB-EVENDIST 160
TABLE 30 PERFORMANCE OF ETSB-SIMTOG AGAINST MINMIN AND BETWEEN GROUPS 161
TABLE 31 PERFORMANCE OF ETSB-SIMTOG METHOD AGAINST MINMIN AND BETWEEN GROUPS 161
TABLE 32 ANOVA RESULTS FOR ETSB-SIMTOG VS. MINMIN AND BETWEEN GROUP CARDINALITY 162
TABLE 33 RESULTS AND PERFORMANCE BY GPMS METHODS 176
TABLE 34 RESULT AND IMPROVEMENT FOR ETB AND ETSB 177
TABLE 35 ANOVA TEST: PRIORITY VS. ETB AND ETSB METHODS 177
TABLE 36 ANOVA TEST: MINMIN, ETB AND ETSB METHODS 178
TABLE 37 SPEEDUP FOR ETB AND ETSB METHODS USING TWO GROUPS 178
TABLE 38 SPEEDUP FOR ETB AND ETSB METHODS USING FOUR GROUPS 179
TABLE 39 SPEEDUP FOR ETB AND ETSB METHODS USING EIGHT GROUPS 179
TABLE 40 GROUPS AGGREGATE MEAN IMPROVEMENT 180
TABLE 41 STANDARD DEVIATION, CORRELATION AND T-TESTS FOR PRIORITY, ETB AND ETSB 180
TABLE 42: ALGORITHM FOR SIMULATING EXECUTION TIMES 214
TABLE 43: ESTIMATING THE JOB SIZE 215
TABLE 44: SAMPLE RESULTS STATISTICS FILE 216
TABLE 45: EXECUTION RESULTS FILE (EXECUTIONRESULTS_MINMIN_4_10000.TXT) 218

List of Figures

FIGURE 1: IMAGE OF THE GRID (TECH4GLOBE 2010) 13
FIGURE 2: THE LAYERED STRUCTURE OF THE GRID (FOSTER, KESSELMAN AND TUECKE 2001) 15
FIGURE 3: DISTRIBUTION OF LARGE PARALLEL COMPUTERS PRODUCED BY VENDORS (SOURCE: TOP500.ORG) 20
FIGURE 4A AND 4B: TWO LEVEL CONTEXT DIAGRAM FOR THE GPMS SYSTEM 90
FIGURE 5: USE CASE DIAGRAM FOR THE GPMS SYSTEM 91
FIGURE 6: ACTIVITY DIAGRAM FOR THE GPMS SYSTEM 93
FIGURE 7: SEQUENCE DIAGRAM FOR THE GPMS SYSTEM 94
FIGURE 8A: CLASS DIAGRAM FOR THE GPMS SYSTEM 96
FIGURE 9B: CLASS DIAGRAM FOR THE GPMS SYSTEM 97
FIGURE 10: A MODEL OF THE GPMS WITH MULTIPLE GROUPS 99
FIGURE 11: A MODEL OF THE GPMS WITH FOUR GROUPS 100
FIGURE 12: FLOWCHART FOR PRIORITY SORTING OF JOBS 106
FIGURE 13: SCHEMATIC DIAGRAM OF THE SYSTEM 116
FIGURE 14: PERCENTAGE AVERAGE AND TOTAL SCHEDULING TIMES FOR MINMIN AND PRIORITY 137
FIGURE 15: SPEEDUP IN MULTIPLES BY PRIORITY OVER MINMIN 137
FIGURE 16: SPEEDUP IN PERCENTAGE BY PRIORITY OVER MINMIN 138
FIGURE 17: TOTAL SCHEDULING TIME OF PRIORITY AND MINMIN WITH INCREASING NUMBER OF JOBS 138
FIGURE 18: TOTAL AND AVERAGE SCHEDULING TIME OF PRIORITY AND MINMIN 139
FIGURE 19: POLYNOMIAL PATTERN OF THE PRIORITY METHODS 139
FIGURE 20: TOTAL AND AVERAGE SCHEDULING TIME FOR ETB-EVENDIST AND MINMIN 149
FIGURE 21: TOTAL AND AVERAGE OF SCHEDULING TIME FOR ETB-SIMTOG AND MINMIN 150
FIGURE 22: SPEEDUP (IN MULTIPLES) OF THE ETB-EVENDIST OVER MINMIN 150
FIGURE 23: SPEEDUP (IN MULTIPLES) OF THE ETB-SIMTOG OVER MINMIN 151
FIGURE 24: SPEEDUP (IN PERCENTAGE) OF THE ETB-EVENDIST OVER THE MINMIN 151
FIGURE 25: SPEEDUP (IN PERCENTAGE) OF THE ETB-SIMTOG OVER THE MINMIN 152
FIGURE 26: PERFORMANCE OF ETB METHODS OVER MINMIN ACROSS GROUPS 152
FIGURE 27: ETB-EVENDIST: IMPROVEMENT ON MINMIN AND ACROSS GROUPS 153
FIGURE 28: ETB-SIMTOG: IMPROVEMENT ON MINMIN AND ACROSS GROUPS 153
FIGURE 29: DECLINING RATE OF IMPROVEMENT BETWEEN GROUPS WITHIN ETB-EVENDIST 154
FIGURE 30: DECLINING RATE OF IMPROVEMENT BETWEEN GROUPS WITHIN ETB-SIMTOG 154
FIGURE 31: TOTAL AND AVERAGE SCHEDULING TIMES OF MINMIN AND ETSB-SIMTOG 162
FIGURE 32: TOTAL AND AVERAGE SCHEDULING TIMES OF MINMIN AND ETSB-SIMTOG BY GROUPS 163
FIGURE 33: SPEEDUP (IN MULTIPLES) BY ETSB-SIMTOG AGAINST MINMIN 163
FIGURE 34: SPEEDUP (IN MULTIPLES) BY ETSB-EVENDIST OVER MINMIN 164
FIGURE 35: SPEEDUP (IN PERCENTAGE) BY ETSB-SIMTOG AGAINST MINMIN 164
FIGURE 36: SPEEDUP (IN PERCENTAGE) BY ETSB-EVENDIST AGAINST MINMIN 165
FIGURE 37: IMPROVEMENT OF ETSB-SIMTOG OVER MINMIN ACROSS GROUPS 165
FIGURE 38: IMPROVEMENT OF ETSB-EVENDIST OVER MINMIN ACROSS GROUPS 166
FIGURE 39: IMPROVEMENT OF ETSB-SIMTOG OVER MINMIN AND BETWEEN GROUPS 166
FIGURE 40: PERFORMANCE IMPROVEMENT OF ETSB-EVENDIST OVER MINMIN AND GROUPS 167
FIGURE 41: RATE OF IMPROVEMENT OF ETSB-SIMTOG ACROSS GROUP CARDINALITY 167
FIGURE 42: RATE OF IMPROVEMENT OF ETSB-EVENDIST ACROSS GROUP CARDINALITY 168
FIGURE 43: SCHEDULING PERFORMANCE BY ALL METHODS WITH INCREASING JOBS 181
FIGURE 44: SCHEDULING PERFORMANCE BY GPMS METHODS WITH INCREASING JOBS 181
FIGURE 45: SPEEDUP BY ETB AND ETSB METHODS USING TWO GROUPS 182
FIGURE 46: IMPROVEMENT ACROSS METHODS USING TWO GROUPS 182

FIGURE 47: SPEEDUP BY ETB AND ETSB METHODS USING FOUR GROUPS 183
FIGURE 48: IMPROVEMENT ACROSS METHODS USING FOUR GROUPS 183
FIGURE 49: SPEEDUP BY ETB AND ETSB METHODS USING EIGHT GROUPS 184
FIGURE 50: IMPROVEMENT ACROSS METHODS USING EIGHT GROUPS 184
FIGURE 51: PERCENTAGE IMPROVEMENT BY ETB AND ETSB METHODS AND BY GROUPS 185
FIGURE 52: IMPROVEMENT BY ETB AND ETSB METHODS ACROSS GROUPS 185
FIGURE 53: IMPROVEMENT COMPARISON BETWEEN THE GPMS METHODS (MULTIPLES) 186
FIGURE 54: IMPROVEMENT COMPARISON BETWEEN THE GPMS METHODS (PERCENTAGE) 186
FIGURE 55: PERCENTAGE AND MEAN SCHEDULING TIME OF THE GPMS METHODS 187
FIGURE 56: AGGREGATE GROUP IMPROVEMENT 187
FIGURE 57: AGGREGATE RATE OF IMPROVEMENT WITH INCREASING GROUP 188

Acronyms

ACO	Ant Colony Optimization
ANOVA	Analysis of Variance
AQUA	Availability-aware QoS Oriented Algorithm
AR	Advanced Reservation
AS	Ant System
BPM	Bank-level Partition Mechanism
CCR	Communication Computation Ratio
CGA	Cellular Genetic Algorithm
CMP	Chip-multiprocessor Systems
CPR	Critical Path Reduction
CPU	Central Processing Unit
DAG	Directed Acrylic Graph
DEQ	Dynamic-Equipartitioning
DFD	Data Flow Diagram
DRAM	Dynamic Random Access Memory
EPU	Effective Processor Utilization
ETB	Execution Time Balanced
ETSB	Execution Time Sorted and Balanced
EvenDist	Evenly Distributed
FCFS	First Come First Serve
FQM	Fair Queuing Memory Scheduler
FR-FCFS	First-Ready-First-Come-First-Serve
GA	Genetic Algorithm
GB	Gigabyte
GHz	Gigahertz
GId	Grid Identifier
GMD	Grid Market Directory
GRBs	Grid Service Brokers
GPMS	Group-based Parallel Multi-scheduler
GRACE	Grid Architecture for Computational Economy
GRAM	Grid Resource Allocation Manager
Grps	Groups
GWF	Grid Workload Format

GWO	Grey Wolf Optimizer
HTC	High-Throughput Computing
IMPS	Integrated Memory Partitioning and Scheduling
IPC	Inter-Process Communication
JVM	Java Virtual Machine
JSP	Job-shop Scheduling Problem
KPB	K-Percent Best
KQML	Knowledge Querying and Manipulation Language
LJFR	Longest Job on Fastest Resource
LLCs	Lower Level Caches
LRU	Least Recently Used
MB	Megabyte
MCP	Memory Channel Partitioning
MCT	Minimum Completion Time
MDS	Metadata Service
MET	Minimum Execution Time
MF	Medium Fast
MI	Million Instructions
MId	Machine Identifier
MMAS	MaxMin Ant System
MIMD	Multiple Instruction Multiple Data
M-LAX	Least-Laxity with Non-pre-emptive Memory
MMAS	MaxMin Ant System
MPI	Message Passing Interface
MPICH	Message Passing Interface for Parallel Computing
NBW	Network Bandwidth
NF	Not Fast
NUMA	Non-uniform Memory Access Architecture
OGSA	Open Grid Service Architecture
OLB	Opportunistic Load Balancing
ORB	Object Request Broker
PAPI	Partially Asynchronous Parallel Implementation
PAR-BS	parallelism-aware Batch Scheduling Algorithm

PDAs	Personal Digital Assistants
PGS	Parallel Genetic Scheduling
PPMS	Priority-based Parallel Multi-scheduler
PREM	Predictable Execution Model
PSO	Particle Swarm Optimization
QoS	Quality of Service
RAM	Random Access Memory
RASA	Resource Aware Scheduling Algorithm
ReqNProc	Requested Number of Processors
RIPS	Runtime Incremental Parallel Scheduling
SA	Simulated Annealing
(SA)	Switching Algorithm
SchedTime	Scheduling Time
SCP	Set Covering Problem
SF	Super Fast
SimTog	Similar Together
SJFR	Shortest Job on Fastest Resource
SMP	Symmetric Multiprocessor or Shared Memory Processor
SP	Speed of Processor
SPEC	Standard Performance Evaluation Corporation
SPI	Synchronous Parallel Implementation
SPMD	Single Program Multiple Data Model
StdDev	Standard Deviation
TotalSchedTime	Total Scheduling Time
TAO	The ACE ORB
TS	Tabu Search
TSP	Travelling Sales Problem
UMA	Uniform Memory Access
UPC	Utility Cache Partitioning
UML	Unified Modeling Language
VF	Very Fast
WCT	Weighted Completion Time
WWG	World Wide Grid

CHAPTER ONE

INTRODUCTION

CHAPTER ONE

INTRODUCTION

1.1 Introduction

This chapter introduces the background to the problem and motivation for the research. It then defines the research question followed by the aims and objectives. This is followed by a description of the methods adopted in achieving the aim and the philosophy behind adopting the method. Then a summary is presented of the findings and research results with a reflection on the research question. Finally, the organization and structure of the thesis is described.

1.2 Background to Problem

Grid computing is growing, gaining more acceptances and making inroads in many spheres of our daily lives. In the same vein, multicore systems are becoming ever more pervasive as hardware computing technology continues to grow in the direction of the Moore's law, although a levelling off is currently apparent. The challenge of scheduling Grid jobs to meet users' requirements and providers' policies in the light of increasing powerful and prevalent computing technology calls for a fundamental and effective rethink. With the advent of multicore computers, scheduling of Grid jobs can be made more effective if scaled to fully utilize the underlying hardware and parallelized to benefit from the gains of the multicores.

Most current Grid scheduling algorithms are sequential in nature and do not consider the inherent benefits in the underlying multicore systems and most focus on scheduling parallel jobs rather than scheduling jobs in parallel. In this research, the phrase "Scheduling jobs in parallel" is used to mean that the actual scheduling task is parallelised whereas "Scheduling parallel jobs" means the scheduling of submitted jobs or tasks such they execute concurrently on various distributed resources.

Scheduling of Grid jobs without considering the underlying multicore hardware in an age characterized by multicore systems does not augur well for the current trend in computing hardware and will constitute the Achilles heel.

Most Grid scheduling algorithms are saddled with overheads incurred in the pre-optimizing computations done before scheduling of jobs. Also, more overheads are incurred when the whole pre-optimizing computations had to be done over again due to arrival and admission of new jobs. Other scheduling problems synonymous with serial scheduling algorithms are the bottlenecks that set in when the number of tasks increases.

Given that sequential applications do not scale with multicore systems nor therefore benefit from parallelism, most current schedulers are rendered unsuitable for today's advances in multicore technology. Hence, as the Grid continues to evolve and grow in tandem with advances in multicore hardware technology, the need to scale Grid job scheduling in-line with the ready benefits of multicore systems cannot be overemphasized.

If the hardware technology of the near future is multicore, then the Grid schedulers of the near future shall be those which utilize the multicores to their benefit. Designing applications to benefit from multicore systems encompasses embracing parallelism. Parallelism enables the optimal use of all available processors and the underlying hardware.

This research aims to develop a method that would exploit multicores through parallelism to enhance Grid scheduling. The result has been the development of the Group-based Parallel Multi-scheduler (GPMS). The GPMS exploits the benefits of multicore systems for Grid scheduling by splitting jobs and machines into paired groups and independently multi-scheduling jobs in parallel from the groups.

1.3 Motivation for undertaking this Work

As multicore computers becomes ever more pervasive in our computing lives and as the Grid continues to grow according to prediction, over reliance on such systems for parallelism does not offer the best option in harnessing the benefits of their multiprocessing capabilities. A means of exploiting parallelism for Grid scheduling is required to tap the full benefit of multicores and place the Grid on a strong footing for the future.

This work was inspired by a number of combined factors. These included:

- Grid computing is an important component in data and compute intensive computing (Geddes 2012) and also provides a backbone in many Cloud systems (Messerschmidt and Hinz 2013). Continued development of new methods to enhance its functioning in an environment of growing data and computational requirements is therefore needed.

- Multicore computers are becoming ubiquitous – the design and development of multicore computers means that single processor systems are being phased out.
- Most Grid scheduling research continues to dwell not on the exploitation of parallelism on multicores in the actual scheduling task but rather on scheduling individual payload tasks in parallel.

The motivation is therefore to delve into the exploitation of parallelism on multicore systems to increase scheduling-throughput in Grid scheduling algorithms.

1.4 Research Question

Most Grid scheduling algorithms are sequential in design and in processing, targeted at addressing issues of QoS and makespan. They do not exploit the opportunities of parallelism as offered by the multicore technology. Since current Grid schedulers are sequential, increased workloads on the Grid could overwhelm the system, create a bottleneck and become its Achilles heel.

Grid computing requires that jobs are submitted by users and executed at remote Grid sites. If the prediction on the growth of the Grid is to become a reality, Grid schedulers would be overwhelmed with the scheduling of millions of jobs at every moment.

Parallelism offers increased speed of processing and optimal utilization of processing components and works best in an environment composed of independent tasks. Grid jobs submitted by different users are diverse and independent and are suitable candidates for parallelisation because such independent tasks offer coarse grain granularity in the scheduling process.

In a multicore environment, each core can be used to do a separate job in parallel. This research was interested in investigating how best the Grid scheduling work could be organised to exploit the benefits offered by such characteristics. A multi-scheduling approach was therefore explored.

Multi-scheduling in this scheme refers to the generation of several independent scheduling instances between independent groups of jobs and groups of machines.

In the light of the above, the research question is:

How can multi-scheduling and parallelism be exploited to take advantage of multicores in order to improve the Grid job scheduling task?

1.5 Aim and Objectives

This research aims to improve scheduling-throughput in Grid scheduling by employing a dynamic approach that exploits parallelism and multi-scheduling to reap the gains of the multicore technology. This aim led to the proposition of the design of the Group-based Parallel Multi-scheduler (GPMS) that is capable of harnessing the benefits of the multicores by exploiting parallelism to leverage the scheduling of Grid jobs.

From the above aim, the following objectives emerged:

- Investigation of current scheduling techniques in Grid and in particular, attempts to exploit parallelism on multicores in Grid scheduling. This objective resulted in the literature review.
- Design of a suitable method to exploit multicore technology through parallelism in the scheduling of Grid tasks. This objective resulted in the group scheduling methodology which included three job grouping methods and two machine grouping methods.
- Design of the multi-scheduler which incorporates suitable multi-scheduling methods. This objective yielded the GPMS.
- Implementation of the three job grouping methods with the two machine grouping methods to exploit parallelism on multicores to enhance scheduling of Grid tasks.
- Design of a suitable test bed and the testing of the group scheduling methods.
- Evaluation of the methods against a widely used non-grouping scheduling algorithms to ascertain the efficiency of the system.
- Discussion of the findings and drawing of conclusion on the work.

1.6 Method

In an attempt to achieve the set goals, the research process was broken down into the following phases:

Literature Review, Definition of Terms and Research Question

This phase dwelt on the review of the literature, related to the research question. Specific areas included: Grid; parallelism; the Grid and parallelism; distributed high-throughput computing systems; parallel scheduling algorithms; Grid scheduling algorithms; gang scheduling; group scheduling and load balancing. This review was carried out to find a gap and grasp knowledge on the task to be engaged in. After the rigorous search in literature, the key terms relating to the research were defined to give a clear context to the work. Then, the research question was formulated.

Design of Grouping Method and Overall Architecture

This phase involved the design of a model to be developed as solution to the problem. This phase brought to existence the visual components of the system, how jobs would flow in and out of the system and what part of the system does what. To bring the design to life, some standard design and modelling tools were used.

A Context Diagram was developed to depict the overall frame of the system with its input and output.

The following UML methods were used to model the system:

- Use Case Diagram that shows how users will use the system.
- Activity Diagram that depicts the activities the system shall perform when fully implemented.
- Interaction Diagram that shows how users will interact with the system.
- Class Diagram that shows the methods and attributes of the system.

A flow-chart was used to describe the logical flow of processes in the system.

Other tools used in the development stage were algorithms and pseudo-code which proved to be very helpful in the coding stage.

The result of this phase was the overall and detailed design of the GPMS.

Implementation of the GPMS

This phase was concerned with bringing the design to life. The Eclipse programming platform was employed for coding. Eclipse was preferred because it offered a very simple platform for programming which was achieved using Java. Multi-threading was used to implement parallelism. This phase also involved testing of the system to ascertain the functionality of the system.

Simulation and Testing

Due to the difficulty in accessing a physical Grid, simulation was employed in the testing process. A Grid environment, with Grid sites composed of several machines which in turn are composed of a number of CPU(s) ranging from 1 to 4 with varying speeds, was simulated for the tests. Also, the execution time of jobs on machines, based on size of the job and the speed of the machine, was simulated and used for the test.

Analysis and Evaluation

This phase involved the use of statistical data analysis tools, querying tools, mathematical formulas and calculations. These were used to analyse and compare test results against results obtained using the ordinary MinMin scheduling algorithm (Ibarra and Kim 1977). The evaluation was carried out to ascertain the efficacy of the method and to appraise the overall success of the research. The outcome of this phase was used to ascertain if the research aim had been achieved and if the research question had been answered.

1.7 Contributions

The main contributions of this research are:

A Group-based Parallel Multi-scheduler has been produced which uses grouping methods to improve the scheduling-throughput in Grid scheduling.

Three novel approaches to group Grid jobs before scheduling in parallel were developed. The three novel methods of grouping Grid jobs are:

- Priority method– this method groups Grid jobs based on job priorities which are in turn computed from the attributes of the jobs or which could be given directly by the users.
- Execution Time Balanced (ETB) – this method uses the execution time of jobs as the attribute to group jobs. It computes the execution time of the jobs based on the size of the job when executed on a standard computer. It then balances the jobs into groups based on the computed execution time.
- Execution Time Sorted and Balanced (ETSB) – This method computes the execution time of jobs, sorts the jobs based on the execution times then balances jobs into groups based on the sorted execution times.

Two novel methods of grouping Grid machines based on the configuration(s) of the machine(s) were developed. The two methods are:

- Similar Together (SimTog) – this method allocates machines with similar characteristics (configurations) into same group.
- Evenly Distributed(EvenDist)– this method distributes all machines fairly equally into the groups based on their configurations.

1.8 Organization of Thesis

This thesis is organized as follows:

Chapter One introduces the work and presents a summary of the entire work: the background; motivation; research question; aims and objectives; method applied in the design; and contributions made.

Chapter Two explores relevant and related literature in Grid computing, scheduling and scheduling algorithms in Grid. The chapter also discussed parallelism, parallel systems, multicore systems, parallel scheduling algorithms, distributed and high throughput computing systems and nature-inspired algorithms.

Chapter Three presents the methodology. It discusses the stages employed in achieving the aims and objectives and discusses the motivation for applying the method.

Chapter Four discusses the design of the Group-based Parallel Multi-scheduler (GPMS) for Grid. It defines the components of the system and the functionality of the components. In a nutshell, Chapter Four serves as the blueprint for the system to be designed. This chapter

brings to life ideas about a solution to the problem. The chapter also describes the algorithms for the methods, simulation, implementation, experimental design and tests of the proposed methods.

Chapter Five discusses the results, analysis and evaluation of results of the GPMS methods (the Priority, the ETB and the ETSB) against the MinMin and also provides a comparative analysis of the three GPMS methods used.

Chapter Six presents a general discussion based on the outcomes in Chapter Five and also presents a brief discussion on shared resources contention among threads.

Chapter Seven compares the GPMS simulator to the GridSim simulation tool and other established systems like Condor. It also relates the method applied in the GPMS to gang scheduling systems and the DIANE scheduler.

Chapter Eight highlights the key points of the thesis, outlines the contributions made to knowledge, draws conclusions and discusses future work.

1.9 Summary

This chapter has provided a background to the problem the research addressed. It introduced the research question and set out the aims and objectives. It then provided an overview of the methods used in achieving the objectives. It also discussed the motivation for the research and presented the contributions made to the field of knowledge. Finally, it provided a glimpse of how the thesis is organized.

The next chapter explores literature in relevant areas of the research.

CHAPTER TWO

LITERATURE REVIEW

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

This chapter explores and discusses literature on relevant areas related to the research. It presents and discusses concepts of Grid, parallelism, multicore systems, parallel scheduling algorithms, Grid scheduling algorithms and also considers work in gang scheduling, grouping and load balancing.

2.2 The Grid

This section introduces various aspects of the Grid. After providing an overview, the general architecture of the Grid is discussed, followed by an exposition of the main types of Grid.

2.2.1 Overview

According to Foster and Kesselman (1999), the Grid is a computing paradigm that promises to change the way complex problems are solved. It was hoped that the Grid would help large-scale aggregation and sharing of computational data and other resources across institutional boundaries otherwise known as virtual organisations. Foster (2000) also observed that properly harnessing the Grid technology will transform various disciplines. These expectations will necessitate the requirement for a computing and scheduling paradigm that meets the expected growth of the Grid (Zhang and Cheng 2006, Etminani and Naghibzadeh 2007, Xiaoyong et al. 2012, and Sajedi and Rabiee 2014). Figure 1 shows the image of the Grid courtesy of (Tech4globe 2010).

Foster and Kesselman (1999) noted that the backbone of the Grid is the already established Internet, the powerful super computers, multiple computing clusters, large scale distributed networks and the connectivity of these resources. The aggregation and integration of these powerful computing systems, clusters, networks and resources, implemented with policies that ensure the delivery of computing services to users' specifications or requirements, is

what represents the Grid. The underlying architecture of the Grid is based on using a set of protocols and heterogeneously distributed Grid resources in order to create Virtual Organizations (VOs). This is implemented on a set of protocols such as OGSA - Open Grid Service Architecture (Foster et al. 2005), using services and middle ware such as Globus (Foster and Kesselman 1997) and implemented upon an enhanced data transfer protocol such as the GridFTP (Allcock et al. 2003).

Figure 1: Image of the Grid (Tech4globe 2010)

The OGSA technologies are service-oriented architectures used by the Grid to provide services to clients using messages. Built from the concept of web services, OGSA is intended to support the creation, termination, management and invocation of stateful, transient Grid services (Bryant 2007). The OGSA framework specifies security, resource provisioning, virtual domains, and the execution environment for other Grid services and API access tools.

GridFTP is a Grid-centric extension to the file transfer protocol with secure, reliable and high- performance data transfer with some added features that meets the concerns of the Grid such as third party control of data transfer, data confidentiality, data integrity and data authentication, stripped data transfer and parallel data transfer (Allcock et al. 2005).

The Globus toolkit is intended to serve as the framework upon which integration of most of the services provided at various layers of the Grid can be accomplished. Globus integrates services between application, middleware and the network. The toolkit provides mechanisms for communication, authentication, network information and data access with the aim of transforming to a system that integrates higher-level services and enables applications to

adapt to the heterogeneous and dynamic meta-computing environment(Foster and Kesselman 1997). The core of Globus toolkit addresses issues of security, resource access, resource management, data movement, and resource discovery.

2.2.2 Architecture of the Grid

The architecture of the Grid is organized in layers with each layer depending on services provided by the succeeding layer as shown in Figure 2 courtesy of (Foster, Kesselman and Tuecke 2001). Each layer is made up of different components and functions and communicates within itself and with the succeeding layer (Laszewski and Mikler 2004). The layers of the Grid are as follows:

- Fabric layer– the fabric layer according to Foster and Kesselman (1999) defines the interface to native resources and implements low-level mechanisms that allow users to access and use resources. It is composed of logical and physical resources. The logical resources include distributed files systems and computer clusters whose access is facilitated by the Grid while physical components includes computational resources, data storage resources, data and networks resources.
- Connectivity layer – this layer defines communication and authentication protocols. The protocols are used for Grid networking and transaction services and also to provide means of identifying Grid users and resources. The connectivity layer includes networking protocols like transport control protocol (TCP) and internet protocol (IP). Other services include the domain name protocol (DNS).
- Resource layer – the resource layer according to Foster et al. (2001) controls access, negotiations, management, monitoring and accounting for Grid resources. It uses the protocols defined in the connectivity layer for these control and management functions.
- Collective layer – this layer oversees and manages the global state and atomic actions of all the resources. It coordinates communications and interactions between Grid resources. It builds upon the services of the lower layers to provide functions like scheduling, brokering, monitoring, diagnostics, data replication and directory services (Netto and Buyya 2010) cited in (Albodour 2011).
- Application layer – the application layer is comprised of users' applications and provides functions that allow for the use of Grid resources. This layer accesses programs, protocols and other services provided by the lower layers.

Figure 2: The layered structure of the Grid (Foster, Kesselman and Tuecke 2001)

2.2.3 Main Types of Grid

Two main types of Grid can be identified: the Data Grid and the Compute Grid.

2.2.3.1 The Compute Grid

The Grid as described by Foster and Kesselman (1999) is a computing paradigm for providing seamless computing services from various heterogeneous compute resources to homes and organizations in a manner analogous to the electricity Grid. It involves the integration and aggregation of different federating computing units to create virtual organizations for the purposes of large-scale sharing and service delivery to meet users' defined QoS requirements (Wieczorek, Hoheiselb and Prodana 2009).

The motivation for the Grid was for the provisioning of computing services on demand by delivering services from various federated and heterogeneous compute resources to homes as utility services like water, gas and electricity. Etminani and Naghibzadeh (2007) and Foster (2000) noted that the compute Grid is aimed at *'solving wide-ranging computational problems in industry, commerce and businesses, engineering and science'*. Foster(2000) also added that the primary target of the Grid is for *'large-scale scientific computations and therefore there was the need for it to scale to leverage large number of resources, enable programs run faster and efficiently and ensure that programs finish correctly with a high degree of reliability and fault tolerance'*. Furthermore, Foster (2000) averred that effectively harnessing the Grid technology will transform various disciplines like high-energy physics, businesses, organisations and the life sciences, enable large-scale aggregation and sharing of

data, computational and other resources across institutional boundaries (otherwise known as virtual organisations) located in disparate geographical regions and provide qualities of service based on policies and protocols. The researcher believes such systems would require a novel, efficient and effective job scheduling mechanism.

With these intimidating promises and sophistications of the Grid, the programming model required in the Grid environments differs fundamentally from traditional serial or sequential execution environments. For instance the need for multiple administrative domains, the heterogeneous nature of resources, diverse policy requirements, quality of services required, stability and performance, and exception handling in highly dynamic environments all place a new demand for Grid programming.

2.2.3.2 Data Grid

The Data Grid was conceived as a service platform designed to provide scalable and optimized management of storage infrastructure and distributed data in the Grid environment (Chervenak et al. 2000). It was conceptualised to address the emergence of large scientific and business applications requiring large amounts of data (in terabytes and petabytes) with diverse requirements, which has brought about the proliferation of various storage devices with specialised capabilities. These various storage devices with varying capabilities have therefore become an integral part of the Grid and need to be managed. How these data and storage facilities can be managed, transferred and replicated is the problem the Data Grid is designed to address (Vazhkudai 2001).

The Data Grid typically uses the Globus Data Grid as a standard platform. The Globus Data Grid architecture is used to define and provide a set of core services that serve as standard to provide access to the diverse storage systems in the Grid environment. A good example of the Data Grid is the European Data Project, set up with the primary aim of developing middleware solutions and test beds that are capable of scaling up to support a novel environment for the global distribution of petabytes of distributed scientific data, are robust in supporting thousands of data centres and processors, and that are capable of managing tens of thousands of multiple users. This has brought about the emergence of fundamental modes of scientific exploration that dissolves the constraints of data-access. The long term goal was the positive impact on future industrial and commercial activities (Segal et al. 2000).

The European Data Grid Project also provides replica management services like the movement and replication of data at high speed from one geographical location to the other, optimization of access to data, management of distributed replicated data and provision of metadata management tools (Cameron et al. 2004).

2.3 Parallelism

Traditionally, computer instructions are written serially and are executed sequentially on single processor systems. Parallel computing is the simultaneous application of multiple computer resources to execute computational problems. This is made possible with the availability of multicores and through the clustering of machines.

Amdahl's law describes a relationship that exists between a serial execution of an algorithm and the parallel execution of the same algorithm using different numbers of processors with the assumption that the algorithm size does not change. While it was observed that for any or many given problems or algorithms there is always a portion of it that can never be parallelized, it was also noted that a **speedup** of processing rate was achievable for every processor added to the system especially if the sequential portion of the algorithm can be parallelized (Amdahl 1967).

Speedup = wall-clock time of serial execution/ wall-clock time of parallel execution

2.3.1 Parallelism and Multicore Systems

Limiting factors on serial computing like transmission speed, miniaturization, need for improved performance, and economic limitations put constraints on the continuous production of serial computers. This trend is traceable to Knight's assertion that limiting factors like size and speed obtainable would always determine the cost / size of computer systems attainable (Kenneth 1966).

Advances in computer hardware technology (owing to Moore's law) has drastically changed the philosophy of computer design from increasing the number of transistors on a chip and increasing clock speed (Moore 1965) to present day multicores (Mellor-Crummey2012, Lin et al. 2009, Peng et al. 2007, Meyer 2006, Geer 2005, Knight 2005,Kalla, Sinharoy and

Tendler 2004, Kenneth 1966). This trend was long predicted by Kenneth and Leon (1975), Hobbs and Theis (1970) and Hollander (1967).

According to Peng et al. (2007) and Gepner and Kowalik (2006), the advances in computing hardware are due largely to the paradigm shift in hardware design. According to Schauer (2008), processor manufacturers have come to embrace the multicore design technology by simply combining two or more individual processors and their caches and controllers in a single silicon-chip, thereby maintaining the system architecture and clock speed and neatly gaining increase in performance.

The length to which the growth of computer technology based on transistors will continue to obey the Moore's law (Moore 1965) or put differently, the extent to which Moore's law would continue to determine the development of silicon chips (transistors) was fascinating and dominated scientific interest and researches in the mid 1960's to early 1970's with the general conclusion that the trend cannot be sustained forever (Hollander 1967, Thurber and Wald 1975). Advances in computer technology were therefore pointed to multichip or multicore processors (Hobbs and Theis 1970, Knight 2005, Peng et al. 2007, and Schauer 2008).

Current technology in computer design has given credence to those predictions as the Moore's law continues to 'level off' and is predicted to gradually die off in 2020 (Eck 2012, and Michiko 2013) to finally pave way for a paradigm shift towards multicore processors (Geer 2005, Meyer 2006, Lin et al. 2009, Mellor-Crummey 2012, and Michiko 2013).

Since every generation of computing technology is identified with its distinct programming platform (Bell 2008), the requirement therefore now is for a programming paradigm that takes advantages of the number of processors (cores), namely parallel programming. Tendulka (2014) noted that for the potential gains of multicore computing to be achieved, a retrospective paradigm shift in software design technology must be embraced.

After a keen observation of the trend of computer evolution from the beginning, characterized by vacuum-tube technologies and accommodation size large enough for many humans to walk in, to today's microprocessor technologies that allow computers to easily fit into a garment pocket, Gordon Bell postulated the Bell's law describing the birth, evolution and eventual death of every computer generation and class based on logic technology evolution

(Bell 2008). A computer class according to Bell is a combination of new platform and 'dominant' programming techniques.

Referring to Bell's Law and Moore's Law, Larus (2009) noted that between the periods 1974 to 2006, the number of transistors on a processors increased from 45 hundred to 291 million – representing an increase of 64, 467 times, while clock speed increased from 2MHz to 2.93GHz – an increase of 1,465 times. This research, if extended to today, will see the numbers jump in millions. This remarkable increase of 40-50% (transistors) per year over the past three decades he contended was underutilized by software and programming codes as the gains were not reflected in processing in much the same way. Noting the paradigm shift from continuous increase in the number of transistors on a processor to the new multicore technology, he opined that one way to gain from the Moore's dividends was to develop codes that execute in parallel and support the design and development of parallel programming languages.

In other related studies by Kessler, Dastgeer and Li (2014) and Catanzaro et al. (2010), it is acknowledged that the propelling idea for advances in computing technology is not just the increase in speed of processing but in improved efficiency and increased throughput. The advent of multicore computing therefore opened up a gap in the software development owing to the fact that execution of serial algorithms on multicore systems impedes performance (Singh and Agrawal 2014, Tendulka 2014, Stone, and Gohara and Shi 2010) and does not optimize the utility of the multicores (Adams et al. 2010). Figure 3 shows the distribution of large parallel computers produced by vendors.

Figure 3: Distribution of large parallel computers produced by vendors (Source: top500.org)

Sadly, these advances in hardware technology are hardly being translated to gains in application design and development. Larus (2009) noted that the gains as a result of advances in hardware technology ('Moore's dividends') are not being fully utilized in software development. He then suggested that codes be developed to execute in parallel. To give more credence to Larus's call, Wang et al. (2007) also noted that multicore systems are not being fully exploited but have the potential for high performance computing if programmed efficiently.

Based on these developments and on a more positive note, recent advances in programming have seen the proliferation of parallel programming languages and applications and most science and engineering platforms are adopting software approaches aimed at utilizing the multicores in their systems to their advantage and fine tuning their applications to enhance the benefits of multicore systems (Ras, Chris and Leo 2011, Ciechanowicz and Kuchen 2010,

LeBlanc and Wrinn 2010, Viry 2010, Nickolls et al. 2008, Ranger et al. 2007, and Stone et al. 2007). Extending these advances and increasing the call for further actions (Jin et al. 2011, Adams et al. 2010, Asanovic et al. 2009, and Chaiken et al. 2008) stated that it is desirable for programming applications to embrace parallelism.

2.3.2 Exploiting Parallelism in Multicore Systems

Sequential algorithms do not scale well with parallel (multicore) systems and single processor systems do not gain from parallel algorithms as well (Bader, Kanade and Madduri 2007, Dolbeau, Bihan and Bodin 2007, Hill and Marty 2008, Nickolls et al. 2008, and Sutter 2005). CPUs (central processing units) have recently been provided with multiple cores and are now capable of processing data in parallel (Lee et al. 2010, and Owens et al. 2007).

Dekel and Sahni (1983) provided an early discussion of parallel algorithms. They presented algorithms for various scheduling problems such as minimizing the number of tardy jobs, job sequencing with deadlines, and minimizing the mean finish time. Maheswaran et al. (1999) also noted that to exploit a given architectural feature of a machine the task's computational requirements must match the machine's advanced capabilities. And Kwiatkowski and Iwaszyn (2010) noted that multicore processors give the opportunity of parallel program execution using the number of available processing units. However, Kwiatkowski and Iwaszyn (2010) opined that when developing programs for multicore computers, consideration should be given to the architecture, parallelism and the number of processors available. For instance, SWAM (Bader, Kanade and Madduri 2007) is a tool which helps in program development and RapidMind (Monteyne 2008) is a tool used in the execution environment. Both tools require knowledge about the processor architecture and parallel programming.

Being the current leading architecture, multicore computers are becoming increasingly pervasive (Bondhugula et. al 2008) and constitute a section of the machines on the Grid. Targeting the parallelism inherent in the multicore systems therefore forms part of the focus for this research.

The GPMS method (Abraham, James and Yaacob 2015a, and Abraham, James and Yaacob 2015b) is intended to exploit parallelism both on the scheduler platform and on the multicore systems that constitute the Grid. The HPC system, on which the GPMS scheduler currently executes, utilises the parallelism provided for in the GPMS method (independent groups that execute scheduling algorithm in parallel / simultaneously). Also, the multicore systems that constitute machines or nodes in the Grid are exploited by scheduling independent jobs directly to them – the scheduler schedules jobs directly to the cores on a machine; machines with two, four or eight cores are allocated two, four or eight independent jobs to execute.

The GPMS method presented in this research is a simulation rather than actual execution and concentration of the research has been on the parallelisation of the scheduler. Hence, the HPC system executes the parallelism inherent in the GPMS at the scheduler level, and also, the multicores that constitute machines in the Grid are exploited in a simulation by scheduling independent jobs to them, thereby improving parallel executions on the Grid machines and improving scheduling throughput.

2.3.3 Multicore Systems and Constraints

Zhuravlev et al. (2012) noted that *'multicore systems have emerged as the dominant architecture choice for modern computing platforms and will most likely continue to be dominant well into the foreseeable future*. *When multicore systems emerged, they executed unmodified scheduling algorithms that were designed for older symmetric multiprocessor systems (SMP). Each core was seen as an isolated processor by the OS; as a result, the SMP scheduler could be used without modifications on multicore systems*. However, to the OS scheduler, this created the illusion that each core in a multicore system was an independent processor. This created a lot of problems. Multicore systems consist of multiple processing cores on a single die and this advancement added a new dimension to the role of the scheduler.

The scheduler in a multicore system carries out both time sharing and space-sharing functions among the threads. Time sharing ensures that threads are scheduled to execute on processors at time intervals while space-sharing entails the actual scheduling of cores to execute the thread chosen to run at the scheduled time in its entirety.

Multicore processors offer tremendous opportunities for parallelism and multi-threaded applications and take advantage of simultaneous thread execution as well as fast inter-thread data sharing. However, as with many systems, multicore systems offer a unique set of challenges. The cores in multicore processor systems are not independent but rather share common resources. The sharing of resources creates contention among threads and this creates problems in multicore systems. Most common shared resources in multicores are the last level cache (L2 or L3), the memory bus or interconnects, Dynamic Random Access Memory (DRAM) controllers and pre-fetchers.

In the Uniform Memory Access (UMA) architecture with multiple Lower Level Caches (LLCs), the LLCs are usually connected via a shared bus to the DRAM controller. This memory bus is a point of contention for threads running simultaneously on the core (Kondo, Sasaki and Nakamura 2007). There is also the DRAM controller which services memory requests that are missed in the LLC. Current DRAM memory controllers were designed for single-threaded access and optimize for data throughput (Rixner et al. 2000). However, the interference among different threads during the scheduling process was not considered in multicore systems. Therefore, when several threads try to access the DRAM controller, these conventional policies gives unpredictable and poor performance (Mutlu and Moscibroda 2008).

Shared resources are managed exclusively in hardware and are thread-unaware; all requests from the various threads running on different cores are seen as if they were all requests from one single source. This means that they do not enforce any kind of fairness or partitioning when different threads use the resources (Zhuravlev et al. 2012).

When multiple cores on a processor share a common resource (cache), this brings about contention between the cores for the shared resources (cache). Contention for the shared cache memory is a major performance bottleneck which also leads to severe and unpredictable performance impact on applications running on the cores. Some researchers have shown that an application can slow down by hundreds of percent if it shares resources with processes running on neighbouring cores relative to running alone (Zhuravlev et al. 2012).

Furthermore, as the number of processor cores per chip increases with new microprocessor generation, the problem caused by shared limited main memory bandwidth is also increased.

Several solutions have been proposed to deal with the negative aspects of multicores and take advantage of the positive aspects. Kondo, Sasaki and Nakamura (2007) used a simulator to evaluate the effect that the shared memory bus can have on the performance of threads in a multicore. They demonstrated when inter-process communication (IPC) between two applications competing for the shared memory bus is reduced, performance can vary dramatically and can cause performance degradation of as much as 60% compared to running solo.

By analyzing the performance impact of mapping processes onto a non-uniform memory access (NUMA) multicore computer with data locality constraints, Majo and Thomas (2011) showed that the operating system alone cannot guarantee good performance in NUMA-multicores if the structure of the memory system and the allocation of physical memory in the system are not considered. The study finds that the benefits of cache contention avoidance can be counteracted if optimization is considered for only data locality and vice versa. They stated that '*the system software must take both data locality and cache contention into account to achieve good performance, and memory management cannot be decoupled from process scheduling*'. They also showed that an architecture-aware process scheduler can greatly increase performance if the operating system is also aware of the memory allocation setup in the system.

Muralidhara et al. (2011) employed application-aware memory channel partitioning (MCP) and integrated memory partitioning and scheduling (IMPS) to reducing inter-application interference in multicore memory systems and improved system throughput by 7.1 percent and 11.1 percent respectively. The MCP maps the data of applications that are likely to severely interfere with each other to different memory channels by partitioning onto separate channels: (i) the data of light (memory non-intensive) and heavy (memory-intensive) applications and (ii) the data of applications with low and high row-buffer locality, while the IMPS prioritizes very light applications in the memory scheduler (since such applications cause negligible interference to others), then uses MCP to reduce interference among the remaining applications.

Liu et al. (2012) applied a practical software approach to effectively eliminate interference in multicore memory without hardware modification by modifying the OS memory

management subsystem to adopt a page-coloring based Bank-level Partition Mechanism (BPM), which allocates specific DRAM Banks to specific cores (threads). The method enabled memory controllers to passively schedule memory requests in a core-cluster (or thread-cluster) way.

The importance of CPU scheduling and resource management in multicore systems is a major concern. Bak et al. (2012) noted that memory-level-interference, caused by simultaneous access to shared main memory by tasks, poses a serious bottleneck to performance. They explored real-time scheduling on jobs adhering to the Predictable Execution Model (PREM) and discovered the least-laxity with non-pre-emptive memory (M-LAX) scheduling policy as the best method. M-LAX schedules accesses to memory in a non-pre-emptive fashion according to least-laxity. The main focus of the study was on PREM jobs - which requires that tasks explicitly indicate during which phases of their execution main memory will be accessed, and during which phases, the application will work with cache-local data.

Zhuravlev et al. (2012) carried out a survey focusing on a subset of proposed solutions to the shared resources contention in multicore systems. Among the multitude of new and exciting work explored, the survey concentrated on solutions that exclusively make use of OS thread-level scheduling to achieve their goals. The solutions include:

Orthogonal CMP-contention minimization techniques

The orthogonal chip-multiprocessor systems (CMP) contention minimization technique involves mapping threads to the cores of the multicore system (Zhuravlev et al. 2012). It was noted that some threads compete less while others compete more aggressively for resources. To mitigate the resource contention with this method, the mapping that gives the best performance is sought. This is done by mapping threads in varying combinations based on their degree of competition for the shared resources. The limitation with this method is that substantial changes are required on the hardware and/or the OS to enforce physical partitioning of resources among threads.

DRAM controller scheduling

DRAM memory is one of the most critical shared resources in a chip multiprocessor. The DRAM memory system in modern computing systems is made up of bank, row, and column. Banks are accessed in parallel, while rows are accessed sequentially. Controllers for DRAM memory systems use a variant of the first-ready-first-come-first-serve (FR-FCFS) policy

(Rixner et al. 2000). FR-FCFS prioritizes memory requests that hit in-the-row buffers associated with DRAM banks over other requests, including older ones.

Two proposed solutions to the DRAM controller scheduling problem are: Fair Queuing Memory scheduler (Nesbit, Laundon and Smith 2007); and Parallelism-aware Batch Scheduling algorithm (PAR-BS) (Mutlu and Moscibroda 2008). The FQM scheduler method keeps a counter called virtual runtime for each thread in each bank which the scheduler increments whenever a memory request of the thread is serviced. FQM prioritizes the thread with the earliest virtual time and balances the progress of each thread in each bank.

The PAR-BS method gives a higher priority to requests from the thread with the shortest stall to minimize the average stall time. The PAR-BS algorithm uses batches to coalesce the oldest requests from a thread in a bank request buffer into units called batches. When a batch is formed, PAR-BS builds a ranking of threads based on their estimated stall time. The thread with the shortest queue of memory requests is heuristically considered to be the thread with the shortest stall time and its requests are serviced preferentially by PAR-BS.

Cache partitioning

The most common replacement policy used in caches is Least Recently Used (LRU) (Suh, Rudolf and Devada 2004, and Kim, Chandra and Solihin 2004). In a single application, the method uses temporal locality by keeping the most recently accessed data in cache. However, when multiple threads share the LLC, the LRU replacement policy treats misses from all competing threads uniformly and allocates cache resources based on their rate of demand (Jaleel et al. 2008). As a result, the performance benefit, that threads with greater cache space enjoy, depends on the memory access pattern and thus varies greatly from thread to thread. Furthermore, it is not a guaranteed that the thread with the most cache space allocation is the one that benefits the most from this space, and by forcing other threads to have less space it can adversely affect the performance of other threads (Qureshi et al. 2006a, and Suh, Rudolf and Devada 2004).

Qureshi and Patt (2006b) proposed the Utility Cache Partitioning (UPC) method which minimizes cache contention among a set of co-running applications. UPC employs a custom monitoring circuit to estimate an application's number of hits and misses, then partitions the cache to minimize the number of cache misses for co-running applications.

Researchers such as Tam et al. (2009), Cho and Jin (2006), Lin et al. (2008), and Zhang, Dwarkadas and Shen (2009) addressed cache contention using a software-based method. The method uses page coloring to partition the cache among applications. A section of the cache is reserved for each application, and the physical memory is allocated in a way that maps the application's cache lines only to the reserved portion.

Software cache partitioning is used to isolate threads that degrade each other's performance. Though this solution holds some promises, it requires nontrivial changes to the virtual memory system and also requires copying of physical memory if the application's cache portion must be reduced or reallocated.

Thread-level scheduling

In the survey, thread-level schedulers were shown to be very effective at mitigating shared resource contention, thus improving performance and predictability. An example of a thread-level scheduler is the contention-aware scheduler (Zhuravlev et al. 2012). Contention-aware schedulers determine which threads are sharing multiple resources and schedule them close together and which threads are sharing minimal resources and schedule them far apart.

Different combinations of threads compete for shared resources at varying degrees, and as such suffer different levels of performance degradation. Using thread-level schedulers to address shared resource contention was found to be attractive because the solution requires no modification to the hardware and minimal changes to the operating system itself.

However, to be truly effective, the schedulers require a workload that consists of both memory-intensive as well as compute-intensive threads in order that co-scheduling threads with complementary resource usage can yield better results compared to contention-unaware schedulers.

Furthermore, contention-aware schedulers are not able to actually eliminate shared resource contention but they can avoid or reduce it. Hence, the study noted that even the best possible thread-to-core mapping may result in high overall contention and performance degradation.

Despite the problems that resource contention introduces in multicore systems, Zhuravlev et al. (2012) conclude that multicore systems present tremendous opportunities for improving

performance of multi-threaded applications. While threads from different applications typically compete for shared resources, threads of the same application can share these resources to their benefit. Threads that share data can also share the same LLC to be more productive. Similarly, such threads can share the prefetching logic and bring data into the cache for each other.

2.3.4 Some Impediments to the Impact of Multicore Systems

The major idea that propels hardware advances was not just the increase in the speed of execution of jobs but also the optimal utilization of processors and increased throughput (Du, Mummoorthy and Jing 2010). The advent of multicore systems therefore created a gap in application design as execution of jobs in sequence in the midst of several processors does not optimize utilization of available processors (Catanzaro et al. 2010). Some of the impediments to legacy systems and applications in the multicore era according to Kumar (2013) include:

Inefficient parallelization – this is an impediment in legacy systems or applications that fail to support multithreading and in some cases too many threads,

Serial bottlenecks – this is mostly common to applications that share a single data source among contending threads, or serialisation of data accessing processes to maintain integrity,

Over dependence on operating system or runtime environment – this arises when too much is handed to the operating system or runtime environment to scale and optimize the application,

Workload imbalance- where the job is unevenly spread to the various cores,

I/O bottlenecks - these occur due to disk I/O blocking,

Inefficient memory management – this is a performance inhibitor caused by the sharing of memory by several CPUs.

To correct or eliminate these impediments, most scientific and engineering platforms have reacted appropriately by embracing and implementing mechanisms that scale utilization of multicores to greater benefit and increasingly issued calls for the design of applications that focus on parallelism in an effort to increase throughput and ensure hardware optimisation

(Kumar 2013, Jin et al. 2011, Ciechanowicz and Kuchen 2010, Stone, Gohara and Shi 2010, and Karp 1987).

The bottom line is that for software applications to gain from the immediate benefits of multicore systems, concerted effort should be made to improve legacy systems and new applications should be developed targeting parallelism. Grid computing will be better leveraged if this method targeting multicore systems finally becomes a reality. This is the driving force for this research as we seek to explore the concept of parallelising Grid scheduling algorithms in order to increase efficiency of Grid scheduling algorithms and increase scheduling-throughput.

2.4 The Grid and Parallelism

Parallelism is a computing paradigm that takes programming away from the traditional serial mode of job processing by employing several computing resources like CPUs to simultaneously execute a given job. According to Foster '*a parallel computer is a set of processors that are able to work cooperatively to solve a computational problem. Such systems include parallel supercomputers that have hundreds or thousands of processors, networks of workstations, multiple-processor workstations, and embedded systems*'.

Most suitable tasks for parallelism are independent tasks that are decomposable and are massively parallelisable. Independent or decomposable tasks are tasks that are easily decomposable into parts and whose computation does not need much communication or sharing of data with other tasks during execution. Such tasks are also referred to as ***embarrassingly parallel*** tasks. On the other hand, tasks that are not easily decomposable and whose data or execution depends heavily on results from other tasks or computations are dependent tasks and are termed non-parallelizable tasks.

On the Grid, tasks are diverse, varied and independent of others as they arrive from different users. Hence they are suitable candidates for parallelisation. Scheduling jobs in the Grid therefore qualifies as one of the most embarrassingly parallelizable tasks.

Some benefits of parallel computing are presented in Table 1.

Table 1 Benefits of parallel computing

No	Benefit	Explanation
1	Save Time and Money	Jobs are done faster, parallel computing can save time and money, though it may cost more to initially acquire parallel computers, but the large time gains far outweigh the initial cost.
2	Solve larger Problems	Parallel computers can solve larger scientific and natural problems so large and complex that it would not be practical to solve them with non-parallel computers – especially problems requiring petaflops of or petabytes of computing resources.
3	Provides Concurrency	It provides concurrency by doing several operations at the same time (Barney 2012).

However, Adams et al. (2010) and Foster et al. (2008) observed that the programming model in Grid environments differs fundamentally from other traditional computing environments. The need for multiple administrative domains, the heterogeneous nature of resources, diverse policy requirements, quality of services, stability and performance, exception handling in highly dynamic environments, the need for scalability to incorporate larger number of resources, the need to enable programs run faster and efficiently and ensure that programs finish correctly with a high degree of reliability and fault tolerance all place heavy demands on Grid programming.

Parallelism is implemented on the Grid using Message Passing Interface (MPI) where tasks use their own local memory during computations and communicate by exchanging messages to and from each other. Ahuja et al. (1986) showed that the Linda programming language is efficient in communication between heterogeneous components by offering facilities for interaction, specification, and dynamic composition of distributed components. They demonstrated that the set of coordinating primitives defined by Linda can be used to implement a Master-Worker parallel scheduler. Fox (2002) implemented message passing in parallel computing and found the code to be executable on all type of architectures, hence declared messaging as the natural universal architecture for the Grid. The MPICH-G2 (Koranis et al. 2003) is a Grid-enabled version of MPI that provides integration with the

Globus Toolkit and provides the same interface of MPI. Mizuno et al. (2003) successfully implemented a system that maintained a pool of pre-spawned threads to handle new tasks and attained concurrency in Grid. Nakada et al. (2003) implemented the Ninf-G GridRPC system which integrates a Grid remote procedure call layer on top of the Globus Toolkit, publishes interfaces and function libraries on the Grid metadata service (MDS) and utilizes Grid Resource Allocation Manager (GRAM) to invoke remote executables.

MapReduce is another programming model that offers support for runtime systems in the processing of large datasets (Dean and Ghemawat 2008). The Map function applies a specific operation to a set of items to produce new items while the reduce function performs aggregation on a set of items. Hence, MapReduce runtime partitions input data and schedules the execution of programs in a large cluster of machines. Another parallel programming implementation in Grid is the Cosmos distributed storage system and the Dryad processing framework developed by Microsoft. It offers DryadLinQ and Scope as declarative programming model on top of the storage and computing infrastructure (Isard et al. 2007). Dryad uses object oriented LinQ query syntax while Scope provides basic operators similar to those of SQL. The release of CUDA (Nickolls et al. 2008) – a parallel programming language has shown that programs can be developed to scale parallelism to leverage the increased number of cores in current computer systems. Another attempt to parallelisation of tasks is the development of Ateji - a parallelism-centric extension to C and C++, Java and other programming languages intended to ease parallel programming constructs and to eradicate some of the common problems of threads in execution (Viry 2010 and Viry 2011).

Despite such efforts, the Grid scheduling community has not given much attention to parallelisation of the actual scheduling process. So far, most Grid scheduling algorithms have concentrated more effort at scheduling the incoming tasks to run concurrently on multiple resources, rather than parallelising the actual scheduling process.

2.5 Distributed and High Throughput Computing (HTC) Systems

A distributed network computing system is an aggregation of networked heterogeneous machines with a set of protocols that enables the sharing of their local resources. Distributed systems consist of multiple autonomous computers, each having its own private memory, communicating through a computer network. Information exchanges in distributed systems are accomplished via message passing. The resource management system is the central component of distributed network computing systems (Hwang, Dongarra and Fox 2013).

Advances in computing have led to an increase in the amount of data being generated. Processing these ever-increasing data in a timely manner has become very challenging; this has led to the emergence of High-Throughput Computing (HTC), a computing paradigm that delivers improved processing deadline by employing data-level parallelism to process data independently on several processing elements using a similar set of operations (Chaudhry et al. 2005, and Lee et al. 2010).

Based on the cost/performance ratio of computer hardware, individuals and small groups now control most powerful computing resources. These owners would only be interested in contributing their resources in a HTC environment only if they are sure that their needs and rights will be addressed and protected. Hence, the challenge facing the HTC environment in order to harness the vast resources available includes: the distributed ownership of computing resources, effective management and exploitation of the available computing resources and how to maximize the amount of resources accessible to its customers (Livny and Raman 1999).

HTC attempts to maximize the number of jobs completed on a daily or longer basis (Tsaregorodtsev, Garonne and Stokes-Ree 2004). The performance goal of HTC technology measures high throughput or the number of tasks completed per unit time. To deliver on this performance goal, HTC systems require parallelism and multicore or many-core processors that can handle large numbers of computing threads per core (Hwang, Dongarra and Fox 2013).

2.5.1 Examples of Distributed Systems

Distributed systems are systems capable of running tasks in parallel on multiple connected resurces and managing their execution. This section discusses parallel and distributed computing platforms. Some examples of distributed computing systems are:

Challenger: A multi-agent system for distributed resources allocation

The challenger (Chavez, Moukas and Maes1997) is a multi-agent system that performs distributed resource (CPU) allocation based on the market model. The system is designed to be robust, adaptive, fault tolerant and to minimize the mean flow time of users jobs. Challenger is composed of local agents with no central control; each agent runs locally on a machine in the network and each agent is capable of assigning tasks originating from the local machine on which it runs and also assigning the local processor.

Challenger agents exhibit the market bidding metaphor; when a job is originated, the local agent advertises the job to all machines requesting for bids in the network (including itself). Information contained in the broadcast includes job id, a priority value and optional information that can be used to estimate how long it will take to complete the job. If an agent is idle when a broadcast is made, it responds by making a bid, which includes the estimated time it will take to complete the job (calculated from the optional information contained in the originating broadcast). If the agent was busy when the broadcast came, it stores the request in a queue in an order of priority. When the agent later becomes free, it retrieves the highest priority request and submits a bid for it. In selecting a match, the originating agent evaluates all the bids and assigns the task to the best bidder (i.e. the agent that returned the lowest estimated completion time). A cancel message is then sent to all agents about the assigned job. On completion of the job, the result is sent to the originating agent.

NetSolve: A network-enabled computational kernel

Netsolve (Casanova and Dongarra 1997) is a network-enabled client–agent–server based application designed to solve scientific problems in a distributed environment. The Netsolve system is an integration of hardware, network resources and computational software packages

in a desktop application. A Netsolve server uses scientific package to provide computational software. Netsolve clients, agents, and servers use TCP/IP sockets for communicaion. Netsolve agents maintain information about resources available in the network. Hence, Netsolve agents have the ability to search for resources in a network, choose the best one available, execute the client request, and then return the answer to the user.

The Netsolve system can best be likened to a computational Grid with a hierarchical organization. An agent may request assistance from other agents in identifying the best resources and scheduling.

Condor: Hunter of idle workstations

Condor (Litzkow, Livny and Mutka 1988) is a distributed high-throughput computing platform for the management of large, distributed and heterogeneous machines and networks. It is designed to exploit idle workstations and it can also be configured to share resources. The Condor distributed platform follows a layered architecture and offers support for both sequential and parallel applications.

Darwin: Resource management for network services

Darwin (Chandra et al. 1998) is a distributed customizable scheduler for creating value added network services. It is designed for the networked environment but can also be adapted for scheduling in non-network nodes. Darwin provides a virtual network to distributed applications.

Darwin is made up of Xena, a resource broker that carries out allocation of resources on a global scale. The system uses a *hierarchical fair service curve scheduling* (H-FSC) algorithm for allocating higher level resources. The H-FSC algorithm is designed to enhance the efficiency of virtual networks for distributed applications.

Ninf: A network enabled server

Ninf (Nakada, Sato and Sekiguchi 1999) is a distributed client–server based network infrastructure for global computing. The system is capable of accessing multiple remote

computers and database servers. The key components of Ninf system include Ninf client interfaces, Ninf Meta-server, and the Ninf remote libraries. The Ninf remote libraries are used to design a global computing application without bothering about the complexities of the underlying system. Ninf applications use Ninf libraries to make requests from the metaserver which contains a directory of Ninf servers. The metaserver respond to remote library calls by allocating resources to appropriate servers by querying the information stored on the servers.

2K: A distributed operating system

2K (Kon et al. 2000a) is an integrated network-centric operating system architecture that aims at mitigating the problems of resource management in heterogeneous networks and allows dynamic adaptability and configuration of component-based distributed applications. 2K is a distributed operating system that provides services across an array of platforms ranging from personal digital assistants (PDAs) to large scale computers. It permits the dynamic instantiation of customized user environments at different locations in the distributed system with mechanisms for proper management of dependencies. 2K is composed of a dynamic TAO (The ACE ORB) (Kon et al. 2000b) which is a reflective CORBA object request broker and an extension of the TAO ORB (object request broker) (Schmidt, and Cleeland 1999). The dynamic TAO ORB creates dynamic environments for applications and moves them across the 2K Grid machines using mobile reconfiguration agents.

Bond: Java distributed agents scheduler

Bond is a distributed Java based object-oriented scheduler for network computing (Boloni and Marinescu 2000). Bond is designed with uniform agent structure and extension mechanism. Bond is agents based (Jun et al. 1999) and uses knowledge querying and manipulation language (KQML) for communication. Bond agents are created as finite state machines with different behaviour in different states. Agents can be checkpointed and migrated by Bond. Agents can discover interface information via an interface discovery service that is accessed via a KQML message. Bond uses two-level scheduler based on a stock market or computational economy approach.

European DataGrid: Global physics data storage and analysis

The European Data Grid Project (Hoschek et al. 2000) is developed to enhance middleware services for distributed analysis of physics data. The system takes advantage of the Globus toolkit as the core middleware. It distributes Petabytes of data in a hierarchical fashion to several sites located worldwide. The system uses global namespaces and special workload distribution facilities to create and access distributed and replicated data. The system integrates data analysis from several hundred scientists in order to have maximum throughput. Information about access and data distribution optimization is enhanced by monitoring users' applications as well as collecting access patterns. Resources are periodically batched and sent to other parts of the Grid. Discovery of resources in the Data Grid is distributed and is done by querying. The scheduler is organized in a hierarchical fashion with an extensible scheduling policy.

Javelin: Java parallel computing

Javelin (Neary et al. 2000) is a system written in Java for Internet-wide parallel computing. The Javelin system uses a distributed approach in scheduling application. The system works like a computational Grid for high-throughput computing.

Javelin is made up of clients that seek resources, hosts that offer resources and brokers that coordinate the allocations between the clients and hosts. Javelin supports a model that decomposes parallel computations into a set of sub-computations. Javelin integrates distributed deterministic work stealing with a distributed deterministic eager scheduler that supports the branch-and-bound model. The model is scalable and fault-tolerant. Another level of fault-tolerance is the implemention of a mechanism that replaces hosts that have failed or retreated.

Nimrod/G: Resource broker and economy Grid

Nimrod/G is a distributed Grid resource broker for managing and steering task farming applications (Buyya, Abramson and Giddy 2000). Nimrod/G is being used as a scheduling

component in Grid Architecture for Computational Economy (GRACE) framework which is based on using economic theories for a Grid resource management system. Nimrod/G has a hierarchical machine organization and uses a computational market model for resource management. It uses the services of other systems such as Globus for resource discovery and dissemination. The scheduling policy is fixed-application-oriented and is driven by user-defined requirements such as deadline and budget.

ProActive

ProActive (Oasis Group 2002) is a Java library which aims to achieve seamless programming for concurrent, parallel, distributed and mobile platforms. ProActive is implemented on the active-object programming model. Each active object controls its own thread and can independently reorder services to incoming method calls. Incoming method calls are stored in a queue of pending requests (called a *service queue*) automatically. Active objects wait for the arrival of a new request when the queue is empty.

Method invocation is used to remotely access active objects. Method calls with active objects are asynchronous with automatic synchronisation. Another communication mechanism is the *group communication* model. Group communication dynamically generates a group of results by triggering method calls on a distributed group of active objects with compatible type.
Migration mechanism is used to move active objects from any Java Virtual Machine (JVM) to another. ProActive is built on top of the standard Java API, and it does not require any modification of the standard Java execution environment, nor does it make use of a special compiler, pre-processor, or modified virtual machine.

2.5.2 Parallel and distributed computing models/offerings

Most companies operating internet-scale services have designed specialized system that suit their need to store and process large data sets. These platforms adopt special methods for processing data in parallel. In these frameworks, the data is staged in compute nodes of clusters or large-scale data centers and the computations are shipped to the data for processing. This section expands the discussion on these systems and also includes discussions on some domain-specific languages designed on top of MapReduce.

MapReduce

MapReduce is a programming model for scalable data processing on large clusters over large data sets (Dean and Ghemawat 2008). Designed to support Google applications, the model is highly scalable and can explore high degrees of parallelism at different job levels. MapReduce is fault tolerant and reliable and provides a framework to implement large parallel system for distributed analysis. A MapReduce computation process can handle terabytes of data on tens of thousands or more client machines. It uses a map function that carries out grouping and a reduce function that performs aggregation. Parallelism is achieved by partitioning the data and processing different partitions concurrently with multiple machines.

However, there are limitations to the model. Olston et al. (2008) noted that '*the map-reduce paradigm is too low-level and rigid, and leads to a great deal of custom user code that is hard to maintain, and reuse'*. Users/programmers are forced to bind their applications to the map-reduce model in order to achieve parallelism. In some other applications, users have to provide implementations for the map and reduce functions. Such custom code is error-prone and hardly reusable. Moreover, complex applications that require multiple stages of map-reduce is difficult to set up. Asking users to implement (multiple) map and reduce functions is like asking them to specifically set out the physical execution plans – which is a difficult undertaking. This often leads to performance degradation by orders of magnitude (Olston et al. 2008).

Hadoop

Hadoop (developed by Yahoo inc.) is a software platform that enables users to write and run applications over vast amount of distributed data. Hadoop (Borthakur 2007) synonymous to Google's MapReduce framework presented as open source. The Hadoop platform uses the Hadoop Distributed File System (HDFS) which is inspired by the Google File System (GFS). HDFS allows data access by Hadoop to take place via a customized distributed storage system built on top of heterogeneous compute nodes. The platform is highly scalable; users can easily scale Hadoop to store and process petabytes of data. Its efficiency is in its ability to

process data with a high degree of parallelism across a large number of distributed machines. It is also reliable as it keeps multiple copies of data for redeployment in case of system failure (Hwang, Dongarra and Fox 2013).

Dryad

Dryad (Isard et al. 2007) is a distributed execution platform for data-parallel applications. A Dryad application is represented as a combination of computational vertices and communication channels which is combined to form a dataflow graph. Dryad executes the application by executing the vertices of the graph on a set of available computers and communicates through files, TCP pipes, and shared-memory FIFOs. The vertices are usually provided by the application developer as sequential programs with no thread creation or locking. Dryad creates parallelism by scheduling vertices (applications) to run simultaneously on multiple CPU cores within a computer or on multiple computers.

Dryad is highly scalable; it can create large distributed, concurrent applications by scheduling the use of computers and their CPUs. It can recover from communication or computer failures.

DryadLINQ

DryadLINQ is built on top of Microsoft's Dryad execution framework to make large-scale parallel distributed cluster computing available to ordinary programmers (Yu et al. 2008). DryadLINQ is composed of two important components: the Dryad distributed execution engine and .NET Language Integrated Query (LINQ). LINQ is designed for users who are familiar with database programming model.

DryadLINQ is a set of language extensions and a corresponding system that automatically and transparently compiles imperative programs in a general-purpose language into distributed computations that execute efficiently on large computing clusters. The goal is to give the programmer the illusion of writing for a single computer and to have the DryadLINQ system deal with the complexities of scheduling, distribution, parallelism and fault-tolerance.

Some domain-specific languages designed on top of MapReduce are:

SawZall

SawZall (Pike et al. 2005) is designed to exploit the parallelism to automate the analysis of very large data sets that span multiple disks and machines distributed over hundreds or even thousands of computers. Designed by engineers in Google Inc, Sawzall is a distributed and parallel data processing system built on top of MapReduce. The Sawzall interpreter runs in two phases; the first phase instantiates processes on many distributed machines, with each instantiation processing one GFS (Google Fie System) file in parallel. The output of this first is used in the second phase – the aggregation phase. The aggregators phase reduces the results to the final output.

The input is initially divided into pieces to be processed separately; these separate pieces may be located on various storage locations. A Sawzall interpreter is then instantiated for each piece of data on the various machines where the data is store on a nearby set.

In each run, more machines will run Sawzall and a smaller fraction will run the aggregator. Due to the aggregator function, the amount of data flowing through the system in each stage is less than at the stage before.

Though the language is interpreted, comparative analysis from experiment have shown that Sawzall is significantly faster than most other languages like Python, Ruby, or Perl and slower than interpreted Java, compiled Java, and compiled C++.

Perfect scaling in the system could see performance almost proportional to the number of machines used. That is every machine would contribute one machine's worth of throughput.

Pig Latin

Pig Latin (Olston 2008) is a language designed to bridge the gap between the declarative style of SQL, and the low-level, procedural style of map-reduce. Designed by engineers at Yahoo Inc., Pig Latin is a dataflow language that uses a nested data model. A Pig Latin program is compiled by the Pig system into a sequence of MapReduce operators that are executed using Hadoop. The system dramatically reduces the time required for the development and execution of data analysis tasks compared to using Hadoop directly.

SCOPE (Structured Computations Optimized for Parallel Execution)

SCOPE is designed for easy and efficient processing of massive amounts of data stored in distributed sequential files and provides efficient query processing functionality (Chaiken et al. 2008). Developed at Microsoft, SCOPE exploits the familiarity of users with relational data and SQL. Scope is designed to run on the Cosmos distributed computing platform for storing and analyzing massive data sets.

SCOPE hides the complexity of the underlying platform and implementation details; thus allowing users to deal only with the task required to solve the problem. The SCOPE compiler and optimizer generates an efficient execution plan and the runtime executes the plan with minimal overhead.

2.6 Parallel Scheduling Algorithms

Parallel computers are made up of collections of processors interconnected in a way to allow a free and parallel coordination of their activities and exchange of data. The processors are located within short distances and are used to solve similar problems (Jada 1992). This contrasts with distributed systems where several processors are distributed over large geographic areas with the goal of exploiting the machines for the purpose of parallel and distributed processing. Parallel scheduling algorithms are algorithms designed to take advantage of parallel computer systems and have been a well researched area.

This section discusses parallel scheduling algothims, which have attracted interest since the early eighties. In the first section, algorithms based on trees, graphs and hypercubes are discussed. In the second section algorithms inspired by nature are discussed. Some of these have been applied to the Grid scheduling problem.

2.6.1 Tree, Graph and Hypercube Parallel Scheduling Algorithms

Dekel and Sahni (1981 and 1983) examined the use of binary trees in the design of efficient parallel algorithms. Targeting the shared memory model of parallel computers and using the

binary tree method, they obtained the complexities and effective processor utilization (EPU) for several parallel scheduling problems. For instance, the researchers used the binary tree method to compute the minimum finish time and minimum mean finish time of jobs. They arrived at a parallel algorithm for minimizing the lateness of jobs and also for minimizing the number of tardy jobs. In the same study, the binary tree method was used to deal with the case of job sequencing with deadlines (this has to do with minimizing the sum of the weights of tardy jobs) and also for minimizing the total cost of the schedule. The binary tree method was also extended in the study to handle the wire routing prolem (Channel Assignment). For all the scenarios they considered in their proofs, they also proved that an effective processor utilization (EPU) can be attained.

An application of trees in parallel scheduling is Tree-Puzzle (Schmidt et al. 2002). The system is a software package for quartet-based maximum-likelihood phylogenetic analysis. The system provides methods for reconstruction, comparison, and testing of trees and models on DNAs as well as protein sequences. As more and more data becomes available in public databases, the runtime of sequential analysis software poses a serious bottleneck. To reduce the wait time of large datasets in the system, the complex aspect of the software that deals with tree reconstruction has been parallelised using message passing to run on clusters of work stations and parallel machines.

Cosnard, Jeannot and Yang (1999), Kwok and Ahmad (1999), Baev, Meleis and Eichenberger (2000), Wu, Shu and Chen (2000), Ranaweera and Agrawal (2001) and Qin and Jiang (2005) modelled the problem of scheduling parallel jobs with a Directed Acrylic Graph (DAG). The DAG models parallel programs with a set of processes (nodes) with dependencies among the nodes. In a DAG, each node represents a task and the directed edges or arcs represent dependencies between the tasks. The nodes in the DAG represent the tasks that are to be executed on the available processors.

Ahmad and Kwok (1995) proposed a low-complexity static scheduling and allocation algorithm for multiprocessor architecture. The method considers communication delays, link contention, message routing and network topology. The method works by first serializing and injecting all the tasks to one processor. Parallel tasks are then ‘bubbled up’ to other processors and are allotted time slots. The edges among the tasks which represent communication links are also scheduled as resources. The method can self-adjust on regular

as well as arbitrary network topologies. The approach was found to be self parallelized, reduces complexity and yielded high speedup. Cosnard, Jeannot and Yang (1999) developed a scheduling algorithm for parameterized DAG; the method derives symbolic linear clusters and then assigns tasks to machines.

Baev, Meleis and Eichenberger (2000) considered two general precedence-constraint scheduling problems in parallel processing. These were minimizing the maximum completion time (makespan) and minimizing the total weighted completion time (WCT). By replacing precedence constraints with release and due dates, they obtained a tight lower bound on makespan and achieved optimal value of up to 90.3% of the time over a synthetic benchmark. They demonstrated that combinatorial algorithm can be a valuable alternative to linear programming in the scheduling of parallel jobs. Qin and Jiang (2005) proposed a dynamic scheduling strategy that provides high reliability for non-pre-emptive, aperiodic real-time jobs. They developed a framework that dynamically schedules real-time parallel jobs dynamically as they arrive at heterogeneous clusters. The approach was shown to make real-time jobs more predictable, reliable and realistic.

Some researches in parallel scheduling distinguished between M-tasks and S-tasks and concentrated on the parallel scheduling of M-tasks. M-task is a task that can be run on a multiple processor computer while S-task is a task that can run only on a single processor computer. Prasanna, Agarwal and Musicus (1994) scheduled M-tasks that are organized in a tree. Ramaswamy, Sapatnekar and Banerjee (1997) used a convex programming model to find the number of processors each M-task will be executed on. The M-tasks are then scheduled to processors using list scheduling algorithm. Before making the final schedule, a balancing act is made between the overall critical path and processor utilization.

Rauber and Runger (1998) used series-parallel (SP) topology to deal with restricted case of graphs. The SP graphs are composed of a set of independent M-tasks that are scheduled by partitioning the processors to disjoint sets and assigning the M-tasks to these processors. Related to this is the work of Subhlok and Vondran (2000), their method focused on scheduling pipelined M-tasks. Also, Radulescu et al. (2001) employed the Critical Path Reduction (CPR) method for scheduling data-parallel task graphs and showed that the method achieves higher speedup compared to other well known existing scheduling algorithms. The CPR method solves the M-task problem in one step as opposed to the two

steps method proposed by Ramaswamy, Sapatnekar and Banerjee (1997). Shu and Wu (1996) proposed the Runtime Incremental Parallel Scheduling (RIPS) method, the method alternates system scheduling activity with the underlying computation during runtime while tasks are incrementally generated and scheduled in parallel. The method targets the Single Program Multiple Data Model (SPMD). The method exploited advanced parallel scheduling techniques to produce low-overhead and high quality load balancing and also adapted efficiently to irregular applications.

Another model for representing the parallel scheduling problem is the hypercube. Ranka, Won and Sahni (1989) developed several examples and described features of a distributed memory Multiple Instruction Multiple Data (MIMD) hypercube multicomputer that can be exploited to obtain efficient parallel program schedules. Using the hypercube model, Cybenko (1989) presented a general approach for studying the convergence rate of diffusion schemes for load balancing. The method analyzes the hypercube network by explicitly computing the eigenstructure of its node adjacent matrix. Using a realistic model of interprocessor communication, the study showed that the deterministic dimension exchange scheme had a better convergence property for the hypercubes.

The use of parallelism to speedup the execution of Branch and Bound (BB) algorithms has also prompted the interest of researchers. This has led to the study of parallel BB algorithms by researchers like Kindervater and Lenstra (1985), Roucairol (1989), Pardalos and Li (1990), Trienekens and de Bruin (1992) and Eckstein (1994).

2.6.2 Nature Inspired Algorithms

Nature inspired algorithms have been applied to scheduling. This section discusses examples of these algorithms. First a variety of previous research is described and then the inherent parallelism in nature inspired algorithms is discussed. Some similar algorithms have been applied directly to the Grid scheduling problem. These are discussed in section 2.7.5 (Nature Inspired Algorithms for Grid Scheduling).

2.6.2.1 Algorithms inspired by nature for scheduling

Nature Inspired Algorithms have become a very active research area because familiar problems are becoming more complex due to size and other dynamics such as changing problem specifications, operating conditions, increasing distribution, decentralisation, robustness, adaptability and improved performance. These have generated new problems that require new solutions because existing methods are not effective. Nature seems to have solved most of its own problems; that is why inspiration is drawn from nature these days and in the foreseeable future.

For instance Liu, Abraham and Hassanien (2010) noted that the dramatic increase in the size of the search space and the need for real-time solutions motivated research ideas into solving scheduling problems using nature-inspired heuristics, while Mirjalili, Mirjalili and Lewis (2014) noted that meta-heuristics have become remarkably common because they possess attractive features such as: simplicity, flexibility, derivation-free mechanism, and local optima avoidance.

Some nature inspired heuristics include Genetic Algorithm (GA), Simulated Annealing (SA), Tabu-Search (TS), Ant Colony Optimisation (ACO), Swarm Optimisation, Cuttlefish algorithm, the Artificial Bee Colony Algorithm, the Firefly Algorithm, the Social Spider Algorithm, the Bat Algorithm, the Strawberry Algorithm, the Plant Propagation Algorithm, the Seed Based Plant Propagation Algorithm, the Grey Wolf Algorithm and many others. In the following paragraphs the Ant Colony Optimisation (ACO) and Grey Wolf Optimisation (GWO) algorithms are discussed.

Dorigo, Di Caro and Gambardella (1999) proposed the Ant Colony Optimization (ACO) meta-heuristic. The algorithm is based on self-reinforcing chemical trails laid by ants while searching for a route (Ridge, Kudenko and Kazakov 2006). In the ACO algorithms, a number of artificial ants cooperatively search for good-quality solutions. Each ant builds a solution by moving through a (finite) sequence of neighbour states (local search) and by publicly available (global) pheromone trails and a priori problem-specific local information. A solution is expressed as a minimum cost (shortest path). High-quality solutions are obtained by the general cooperation among all the agents of the colony.

The ant algorithm has been successfully applied by Colorni et al. (1994) to the job-shop scheduling problem (JSP). The job-shop scheduling problem assigns machines so that the

maximum of the completion times of all operations is minimized and no two jobs are processed at the same time on the same machine. When applied to problems of dimensions up to 15 machines and 15 jobs, the solutions were always within 10% of the optimal value (Colorni et al. 1994 and Dorigo, Maniezzo and Colorni 1996).

Inspired by the activities and types of grey wolves, Mirjalili, Mirjalili and Lewis (2014) proposed the Grey Wolf Optimizer (GWO) meta-heuristic. The GWO algorithm mimics the leadership hierarchy and hunting techniques of grey wolves in nature. Four types of grey wolves such as alpha, beta, delta, and omega are employed for simulating the leadership hierarchy. In addition, the three main steps of grey wolf hunting such as: searching for prey, encircling prey, and attacking prey, are implemented. The results show that the GWO algorithm is able to provide very competitive results compared to other well-known meta-heuristics.

2.6.2.2 Parallelism inherent in nature inspired heuristics

Some of the common characteristics of nature's heuristics are the close resemblance to a phenomenon existing in nature, nondeterministic nature, presence of ***implicitly parallel structure,*** and adaptability (Abraham, Buyya and Nath 2000). Ridge, Kudenko and Kazakov (2005) noted that natural systems on which nature inspired algorithms are based, possess many desirable properties that makes them good candidates for parallelism. These properties include, large numbers of relatively simple participants (ants, wolves, bees, birds, fish), completely decentralised, operate in parallel and asynchronously, use of relatively simple signals and their desired functionality emerges from the interactions of their participants.

Generally, parallelism is inherent in systems with distinct and decomposable operations, tasks with high degree of independent or data with low relationships. These features are very prominent with nature-inspired heuristics. For instance, the nature of search for solution in nature-inspired heuristics provides a great opportunity for parallelism. Search for solution in meta-heuristic algorithms are based on global and local searches. Liu, Abraham and Hassanien (2010) noted that the focus is shifting to nature-inspired meta-heuristics because of the sound exploration ability of both global and local optimal solutions. Referred to as exploration and exploitation respectively, the exploration phase refers to the process of investigating the promising area(s) of the search space as broadly as possible (globally) while

the exploitation phases involves the (local) search around the promising regions obtained in the exploration phase (Mirjalili, Mirjalili and Lewis 2014).

Both the local and global search methods are decomposable and present opportunities for parallelism. Secondly, the global solution relies on the local search solutions which are performed by individual ants in the colony or bird (or fish) in the swarm. Since each bird, fish, or ant (as the case maybe) can act independently or concurrently, this again provides another opportunity for parallelisation.

For example the very nature of ACO algorithms posseses features of parallelism. In particular, many parallel models used in other population-based algorithms can be easily adapted to the ACO structure (e.g. migration and diffusion models adopted in the field of parallel genetic algorithms) (Campanini et al. 1994, and Dorigo and Maniezzo 1993). Early experiments with parallel versions of ant systems for the travelling salesman problem (TSP) approached the problem by attributing one processing unit to each ant. The limitations with this method is the communication overhead due to ants spending most of their time communicating to other ants the modifications they made to pheromone trails. Bolondi and Bondanza (1993) obtained better results on a coarse grained parallel network of 16 transputers by dividing the colony into sub-colonies based on the number of available processors. The sub-colony acts as a complete colony and therefore implements a standard Ant System (AS) algorithm. After the sub-colonies has completed the iteration of the algorithm, a concurrent update of the pheromone trails is carried out via a hierarchical process that collects the information about the tours of all the ants in all the sub-colonies and then broadcasts this information to all the processors. The method recorded a speed-up that was nearly linear when increasing the number of processors, and this behaviour did not change significantly for increasing problem dimensions.

Bullnheimer, Kotsis, and Strauss (1997) proposed two coarse-grained parallel versions of AS: the Synchronous Parallel Implementation (SPI) and Partially Asynchronous Parallel Implementation (PAPI). The SPI is related in implementation to the one implemented by Bolondi and Bondanza (1993) while the PAPI exchanges pheromone information among subcolonies for every fixed number of iterations done by each sub-colony. The findings show a better performance of the PAPI approach with respect to running time and speed-up which

was due to the reduced communication as a result of less frequent exchange of pheromone trail information.

Stutzle (1998) presents computational results for the execution of parallel independent runs on up to 10 processors of his MaxMin Ant System (MMAS) algorithm (Stutzle and Hoos 1997a, Stutzle and Hoos 1997b). The results showed that the performance of MMAS grows with the number of processors. This is due to the parallelism inherent with the ACO algorithms.

Kwok and Ahmad (1999) proposed a Parallel Genetic Scheduling (PGS) algorithm that relies on two powerful genetic operators: the order crossover and mutation. PGS is a parallel algorithm which encodes the scheduling list as chromosomes and uses that to generate high quality solution. The PGS outperformed two heuristics best known for performance and time complexity. The PGS also attained optimal solution for more than half of the test cases. This demonstrates the parallelism inherent in nature's heuristics.

2.7 Grid Scheduling Algorithms

This section introduces the need for Grid scheduling algorithms as well as presenting and discussing some existing Grid scheduling algorithms.

The Grid computing environment requires that jobs are submitted by users and executed at remote Grid sites. Scheduling on the Grid differs from traditional scheduling on computer systems and clusters (Tchernyk et al. 2006). Scheduling on computer systems and clusters is aimed at achieving optimal utilisation of resources and meeting the conflicting need of processes on limited resources. On the Grid there are several machines available and several Grid sites. The processes or tasks are not in contention for limited resources. Hence, the scheduling is aimed to meet the diverse QoS requirement of jobs from different users. Discussing the differences in application scheduling between clusters (and by extension traditional computing systems) and the Grid, Buyya and Murshed (2002) noted that '*the scheduler in clusters aims at improving overall performance and system utility while schedulers in Grid systems aims at improving performance of applications inorder to meet end user requirements'.* This requires reliability of the hardware and software, efficiency in

time consumption and effectiveness in the utilization of resources as well as increased throughput.

Finding the optimal schedule is an NP-complete problem and so heuristics are typically used. Alternatively, non-deterministic algorithms such as genetic algorithms can be used. However if the scheduling algorithm becomes too complex, the benefits of obtaining an optimal solution is outweighed by the time it takes to schedule.

The importance of understanding the Grid and the concept of scheduling cannot be over emphasized. Feitelson, Rudolph and Schwiegelshohn (2004) highlighted the importance of a Grid scheduler and warned that the whole Grid system can fail should the scheduler fail. More recently Prajapati and Shah (2014) presented work to give a concise understanding of the concept. The work classified Grid scheduling algorithms and discussed methodologies used in evaluating Grid scheduling algorithms. For Grid computing to meet the requirement for large-scale international and global resource sharing and grow in the right direction, an effective and efficient scheduling algorithm will be required to facilitate throughput and enhance scalability (Sajedi and Rabiee 2014, Tang et al. 2012, Etminani and Naghibzadeh 2007, and Zhang and Cheng 2006). The scalability requirement also demands that Grid scheduling algorithms are dynamic and reactive to the trend in hardware computing technology by exploiting the benefits the technology brings (Klusacek 2008).

Inventions and advances necessitate changes, hence scheduling has transformed in several ways owing to the evolution of the computers to effectively control the conflicting demands from various processes for the limited CPU, memory and I/O resources. The Grid is a specific environment and requires a specific scheduling approach. The scheduling of Grid jobs has generated much interest and has continued to occupy the centre stage in recent research (Yu and Yu 2009).

In the following sections, Grid scheduling algorithms are discussed. Firstly the classical algorithms are discussed, and then fusion and enhancement of such algorithms are discussed. Next QoS focussed algorithms are considered, followed by adaptive Grid scheduling algorithms and algorithms based on nature. A selected, representative list of Grid scheduling algorithms is provided in Appendix C.

2.7.1 Classical Grid Scheduling Algorithms

Scheduling in the Grid can be carried out in immediate mode or batch mode. Immediate mode is when a job is assigned to a machine as it arrives and batch mode is when a number of jobs are batched and scheduled together (Maheswaran et al. 1999). Batch mode algorithms include the MinMin and MaxMin algorithms introduced by Ibarra and Kim (1977). The MinMin algorithm computes the completion time for all jobs on all machines then iteratively assigns the job with the minimum completion time to the processor that can complete the job the earliest.

The MaxMin algorithm applies a similar principle to MinMin by computing the completion time for all the jobs on all the processors but the jobs with the maximum completion time is assigned to the processor that can complete the job earliest. Another batch mode algorithm is the Sufferage algorithm introduced by Maheswaran et al. (1999). The Sufferage heuristic is based on the idea that better mappings can be generated by assigning a machine to a task that would 'suffer' most in terms of expected completion time if that particular machine is not assigned to it. Algorithms for immediate mode include: the traditional First Come First Serve (FCFS); Easy-Backfill, which optimises FCFS by allowing jobs to jump the queue where they can fit a gap which otherwise would be left empty due to requirements of the next job in line. Opportunistic Load Balancing (OLB), where a task is assigned to the machine that becomes ready next, without considering the execution time of the task onto that machine; minimum execution time (MET) where the job with minimum execution time is selected next; minimum completion time (MCT) where the job with minimum completion time is selected next; and k-percent best (KPB). The k-percent best (KPB) heuristic considers only a subset of machines while mapping a task. The subset is formed by picking the k-percent best machines based on the execution times for the task. The task is assigned to a machine that provides the earliest completion time in the subset (Maheswaran 1999).

Maheswaran et al. (1999) compared new and previously proposed dynamic matching and scheduling heuristics for mapping independent tasks onto heterogeneous computing systems under a variety of simulated computational environments. Five immediate mode heuristics and three batch mode heuristics were studied. For immediate mode they investigated opportunistic load balancing (OLB), minimum execution time (MET), minimum completion

time (MCT) and k percent best (KPB). For batch mode they considered MinMin, MaxMin and Sufferage. Sufferage was a new algorithm proposed by the researchers. The authors showed that the choice of dynamic scheduling heuristic in a heterogeneous environment depends on parameters such as heterogeneity characteristics of task and machine as well as the arrival rate of tasks.

2.7.2 Fusion and Enhancement of the Classical Algorithm

Freund et al. developed SmartNet (1996, 1998) which is a resource scheduling system for distributed computing environments. The work focused on the benefits that can be achieved when the scheduling system considers both computer availability and the performance of each task on the computer. The system requires jobs to be broken down into tasks and also requires estimates of execution time of tasks. It collects and uses data on jobs, task, machines and networks in order to tune the scheduling outcomes. The system uses various scheduling algorithms to attempt to assign tasks to the computer that will run that task best. The work is interesting as it was one of the first to consider detail of computer, task, job and network characteristics in scheduling. SmartNet implemented the MinMin and MaxMin algorithm introduced by Ibarra and Kim (1977). The SmartNet approach showed improvement over simple load balancing.

Other researchers have used various combinations or adjustments to these methods to improve the schedule. Lawson and Smirni (2002) employed greedy scheduling algorithms and conservative backfilling to schedule parallel jobs in a heterogeneous multi-site environment. Greedy scheduling algorithms are algorithms that consider the immediate or current best solution without recourse to the long term implications. Feitelson, Rudolp and Scwiegelshohn (2005) presented a status report extending surveys of scheduling parallel jobs from supercomputers to clusters and Grid. Zhang, Albert and Mingzeng (2006) employed greedy-heuristics adaptive resource selection strategies and the conservative and easy back-filling algorithm to schedule parallel tasks.

Venugopal and Buyya (2008) proposed a heuristic approach based on the Set Covering Problem (SCP) to schedule distributed data on the Grid. The approach mapped jobs to storage resources on the Grid and then mapped storage resources to datasets required by jobs and scheduled a set of jobs to a set of compute resources using the MinMin heuristics or

Sufferage algorithm. The experiment showed that the method when combined with Exhaustive search heuristic performed better than the other four heuristics but noted that Exhaustive search may search through large spaces for jobs requiring large datasets. They also noted that there was no gain in performance when combining the method with MinMin and the Sufferage algorithm (Venugopal and Buyya 2008).

2.7.3 QoS-Focused Algorithms

More latterly, research has focused heavily on Quality of Service (QoS) paying attention to user requirements.

Buyya, Abramson, and Giddy (2000) employed a resource reservation mechanism to schedule jobs on the Grid. Their method supported resource reservation request scheduling models implemented on First Come First Serve (FCFS) and Easy-backfilling.Buyya, Abramson, and Giddy (2000) took cognizance of the heterogeneous nature of the Grid, user defined QoS, and resource owner services availability to design the Nimrod-G resource broker and scheduler. Using economic models of demand and supply to represent resource management and allocation issues, they were able to regulate supply and demand as scheduling activities on the Grid. The model implemented Grid provider services and consumer demand based on some common economic principle of supply and demand. The model considered three key players in the Grid marketplace viz: Grid Service Providers (GSPs) that represent the producers; Grid Service Brokers (GRBs) that represent brokers; and Grid Market Directory (GMD) which is the medium through which the two players interact. The model was subsequently implemented on GRACE - to provide an economic incentive for resource owners to share their resources and resource users to trade-off between deadlines and budgets (time and cost). Although this model has become widely accepted in Grid, certain concerns were neglected. These were: the variation of users need; the human need; social need; changing technology; and other dynamics. Such concerns conspire to make it impossible for the Grid to solely rely on economic models based on principles of supply and demand.

To adequately schedule jobs on the GRID, He, Xian-He and Laszewski (2003) suggest that consideration be given to two new concepts:

- How to calculate the computation time for the job on the non-dedicated network.
- The quality of services required by the user.

With those goals as a guide and based on the general adaptive scheduling heuristics, He, Xian-He and Laszewski (2003) designed the QoS guided task scheduling algorithm for Grid computing and recorded a significant performance gain in different applications. Armed with that success, the researchers extended the algorithm and designed the QoS guided MinMin scheduling algorithm for the Grid. The algorithm provides a match between the QoS requirements of a user's job to the QoS provisions available from Grid service providers and provides an estimate of the completion time of the job. The algorithm favours smaller jobs by allocating smaller jobs to faster Grid resources and allows bigger jobs to starve. This heuristic does not fully take the QoS requirements of user jobs into consideration as QoS in this heuristic was treated more as the bandwidth requirements of the job and not the speed of the CPU.

Zhoujun, Zhigang and Zhenhua (2010) designed a cloud based dynamic service evaluation system with a method to cluster all services with similar QoS and then a dynamic meta-task scheduling algorithm that provided services to users based on QoS and clusters(Zhoujun, Zhigang and Zhenhua 2010). The drawback with this algorithm is the assumption that all Grid resources provided the same QoS. Secondly, the heuristic is more concerned about makespan and cost reduction but overlooked the need to accommodate for the future growth.

Due to the heterogeneous nature of the Grid there is a strong probability that at most times, some Grid resources may not be available. This may be as a result of break downs, network failures, local user policy, software failures, system malfunctions, management decisions or other locally based factors. Scheduling jobs to resources whose availability is not certain introduces yet another dimension to Grid scheduling. To ensure reliability and user satisfaction, solutions to such problems must include a high degree of certainty that the Grid resource is available before scheduling jobs to it. Based on this concern, Agarwal and Kumar (2011) proposed the (AQuA) algorithm that schedule jobs to Grid resources based on:

- A high probability of the availability of the resource at the time of scheduling.
- A satisfaction of the QoS required by the job.

This heuristic modeled the bandwidth requirements of job as the QoS of a network and availability of requirement as QoS of compute resource. The QoS requirement is then implemented using the MinMin heuristics. The algorithm reduces the makespan of jobs when compared against the QoS guided MinMin heuristics.

Caminero et al. (2011) proposed a network-aware multi-domain meta-scheduling strategy based on peer-to-peer techniques. The method coordinates the interaction of resources between administrative domains especially when performing meta-scheduling of jobs, job migration or monitoring of jobs. Using the routing indices method of peer to peer systems to forward queries (Crespo and Garcia-Molina 2002), the method considers forwarding of queries to neighbouring peers that are more likely to have the computing resources for a users' job within the domain before others to avoid random sending and flooding of the network. It also takes into account the characteristics of the network when performing meta-scheduling. It considers communication and queries between domains and also offers scalability. The method was implemented on the GNB (Grid Network Broker) and recorded better success rate of jobs and better latencies with less queries per job. They also recorded less overhead which makes the system scalable.

Shah, Mahmood and Oxley (2011) explored the dynamic nature of incoming jobs for scheduling. In a related study, Shah et al. (2012) proposed the QoS based performance evaluation of Grid scheduling algorithms in which they carried out a comparative performance analysis of their job scheduling algorithm along with other algorithms based on QoS parameters like waiting time, turnaround time, response time, total completion time, bounded slowdown and stretch time. They confirmed from evaluation that their algorithm possesses a high degree of performance efficiency and scalability in Grid.

Albodour, James and Yaacob (2012 and 2014) proposed the BGQOS, a QoS model for business-oriented and commercial applications on Grid computing systems. BGQoS allows Grid Resource Consumers (GRCs) to request specific QoS requirements from Grid Resource Providers (GRPs) for their resources to be utilised. BGQoS supports the dynamic calculation of QoS parameters such as resource reliability. This increases the accuracy of meeting the GRC's requirements. GRPs are capable of advertising their resources, their capabilities, their usage policies and availability both locally and globally. This created a flexible model that could be carried across domains without altering the core operations and which could easily

be expanded in order to accommodate different types of GRC, resources and applications. Methods that monitor and reallocate jobs are used to ensure that QoS targets are met.

Xiao and Dongbo (2014) proposed a Multi-Scheme Co-Scheduling Framework (MSCSF) to provide enhanced deadline-guarantees in heterogeneous environments. The works integrates multiple co-scheduling schemes and quantitatively evaluates the deadline of each co-scheduling scheme. The system then selects the best scheduling scheme for real-time applications at run time. Experimental results show that it can provide enhanced deadline-guarantee. Chen, Li and Wang (2014) proposed a model to support the parallel strict resource reservation request scheduling model and algorithm. The method supported resource reservation request scheduling models implemented on First Come First Serve (FCFS) and Easy-backfilling. They presented the FCFS and Easy backfilling analysis of two important parallel algorithms based on job bounded slowdown factor and the success rate of Advanced Reservation (AR). Simulation results of the combined four methods showed that the easy backfilling method + first-fit algorithm can ensure the QoS of AR jobs while taking into account the performance of non-AR jobs.

2.7.4 Adaptive Grid Scheduling Algorithms

Some researchers have attempted to use characteristics of jobs to drive adaptive scheduling methods in attempt to gain better results. Classical heuristics favour one set of jobs to the detriment of the other set. For instance MinMin favours small jobs, while MaxMin favours large jobs.

To ensure that one set of jobs do not suffer at the expense of the other and to address the problems of starvation of large jobs inherent in the MinMin heuristic, Etminani and Naghibzadeh (2007) designed a new selective scheduling algorithm to select at each decision point, the best algorithm between MinMin and MaxMin according to length of tasks in the batch. For instance if there is a prevalence of long tasks in the remaining tasks in a batch, the MaxMin would be chosen, if there is a prevalent of short tasks, the choice would be MinMin(Etminani and Naghibzadeh 2007). Parsa and Entezari-Maleki (2009) also proposed the implementation of RASA (Resource Aware Scheduling Algorithm), an algorithm that combines both MinMin and Max-Min heuristics; alternatively executing both heuristics in strict order. Experimental analysis led them to conclude that the heuristic was better than

both MinMin and MaxMin but the study did not take into consideration the dynamics of the Grid. RASA only concentrated on the current number of jobs to be scheduled and which heuristics to apply at any given time. For instance if the number of jobs in the queue was odd- then apply MinMin heuristic, if the number of jobs was even – then apply MaxMin heuristics. The MinMin is used to favour smaller tasks while the MaxMin is used to favour large jobs.

Caminero et al. (2007) noted the high variability in Grid environment and how it affects desirability of QoS and also pointed out the difficulty in achieving that desire. They then went ahead to propose the autonomic network-aware Grid scheduling architecture as a solution. The system was capable of making decisions based on its current network status and adapts itself to changes. It also incorporated a model for predicting the latencies in a network and in CPU which allows the architecture to exhibit the autonomic behaviour. This work was more focused on the architecture of the Grid in order to satisfy some QoS requirements than the scheduling of Grid jobs in parallel.

Liang et al. (2013) used behavioural clustering of execution time to establish a pattern for users' jobs and used that to improve accuracy of overall job execution times. The approach implemented a method to evaluate execution time estimation for parallel jobs based on user behaviours clustering for execution time estimation by exploring the job similarities and revealing the user submission patterns. The result showed that the approach improved the accuracy of job execution time estimation up to 5.6 % and the time for performing the computation was reduced by 3.8%. Khan, Kalim and MostafaAbd-El-Barr (2014) used a non-FCFS policy to schedule parallel jobs by monitoring incoming jobs and their resource requirements to make scheduling decisions based on the backfilling algorithm. The authors used task partitioning and load balancing to schedule data parallel tasks. Wang et al. (2014) implemented the comprehensive performance tuning framework to initially schedule jobs to resources, and later tune certain parameters for another round of job scheduling.

2.7.5 Nature Inspired Algorithms for Grid Scheduling

Using biological theories of natural selection, survival of the fittest and how populations evolve and adapt, Abraham, Buyya and Nath (2000) proposed the use of the Genetic Algorithm (GA), Simulated Annealing (SA - originally by Osman and Potts 1989) and Tabu-

Search (TS - originally by Widmer and Hertz 1989) heuristics. They claimed that the GA can provide solutions to real world problems if properly programmed. According to the researchers, GA is adaptive and can be used to solve optimization problems based on the genetic process of biological organisms. They stated that GA searches are neither constrained by the continuity function nor the existence of a derivative function. Hence they declared that GA can easily adapt to the principle of natural selection and survival of the fittest.

The researchers described the SA as an algorithm that exploits the analogy of the annealing process (that enables metals to cool and freeze into a minimum energy crystalline structure) and the search for a minimum in a more general system. They stated that the SA has the ability to avoid being trapped at local minima.

Furthermore, the researchers stated that the TS was a meta-strategy known for guiding other known heuristics towards overcoming local optimality by repeatedly making moves from a set of solutions to other sets with the aim of efficiently achieving optimal solutions by the evaluation of some objective functions. They then went further to propose the hybridization of GA-SA as a scheduling algorithm for the Grid; arguing that it has the potential to inherit properties of both GA and SA to yield optimum. The study equally proposed the Hybrid GA-TS (combination of GA and Tabu-Search), arguing that the combination of GA-TS will makes the result robust but the effectiveness of this is yet to be tested. A simulated experiment was carried out for only the GA algorithm with a finite number of resources (three computing resources) and thirteen jobs with an assumption that the processing speed of the resources, the cycles per unit time and the job length (processing requirements in cycles) are known. The simulation showed that all the resources were efficiently utilized and the jobs completed in minimum time. But only three resources and thirteen jobs is too minuscule to consider generalizing for the entire Grid.

Sabin et al. (2003) explored the use of queues to model performance dynamics of resources to schedule independent tasks based on deadline and afterwards applied a neural model to schedule subtasks. Carretero and Xhafa (2006) implemented the GA for job scheduling on computational Grids that optimizes the makespan and the total flowtime. The aim is to obtain an efficient scheduler that is capable of allocating a large number of jobs originated from large scale applications to Grid resources. The results recorded a fast reduction of makespan,

showed the robustness of the GA implementation and improvement in performance over the MinMin, thus making the GA a scheduler of practical interest for Grid environments.

Liu, Abraham and Hassanien (2010) introduced a fuzzy approach based on Particle Swarm Optimization (PSO) for scheduling jobs on computational Grids. The particle swarm algorithm is inspired by social behaviour patterns of organisms that live and interact within large groups. In particular, it incorporates swarming behaviours observed in flocks of birds, schools of fish, or swarms of bees, and even human social behaviour, from which the Swarm Intelligence (SI) paradigm emerged. The fuzzy scheme based on discrete PSO extends the vectors of fuzzy matrices to represent the position and velocity of the particles for computational Grid job scheduling. The fuzzy approach dynamically generates an optimal schedule so as to complete the tasks within a minimum period of time as well as utilizing the resources in an efficient way. As an algorithm, its main strength is its fast convergence, which compares favourably with many global optimization algorithms.

2.8 Parallelisation of the Grid Scheduling Task

This section discusses the problems with current Grid schedulers and makes a case for the parallelisation of Grid schedulers.

2.8.1 Problems with Current Scheduling Algorithms

The algorithms described in section 2.7 map specific job(s) to specific Grid machine(s) based on some factors such as job, machine or network characteristics, (QoS) criteria and policies. They are based on overall performance in terms of scheduling and completing the whole task set or on providing improved quality of service to users. The algorithms described so far do not focus on the parallelisation of the Grid scheduling task but focus instead on reducing overall makespan. Thus, they do not concentrate on improving the efficiency of the scheduler in terms of how long the scheduling task takes or how much jobs are scheduled in a given time; hence, they do not utilize the underlying hardware for full benefits of parallelism. Even though the nature inspired algorithms have inherent parallelism, there has, even in this area, been little focus on the effect of parallelisation of the Grid scheduling task itself.

Gupta, Tucker and Urushibara (1991), Ryoo et al. (2008), Agarwal and Kumar (2011), and Xiao and Dongbo (2014) have shown that many such algorithms cannot be optimal in

scheduling as they lack the ability to leverage Grid scheduling. Schwiegelshohn et al. (2010) noted that to adequately harness the power and functionalities of the Grid and leverage Grid scheduling in tandem with the dynamics, scalability and growth in computing, a more drastic, scalable and dynamic approach will be required.

2.8.2 Parallelisation of the Grid Scheduling Algorithms

For Grid scheduling to gain from the advances in hardware technology, meet its protracted growth and the challenges of the future. It is imperative for a paradigm shift in software programming model (McCool 2008). This is because sequential programs do not scale with multicore systems nor benefit from parallelism due to performance limitations (Gurudutt-Kumar 2013, Hill and Marty 2008, Nickolls et al. 2008, Bader and Cong 2011, Dolbeau, Rihan and Bodin 2007, and Sutter 2005).

As mentioned earlier, most work on Grid scheduling has concentrated on creating a parallel schedule for executing the jobs that are input to the scheduler. Less attention has been paid by researchers to the actual parallelisation of the scheduling task. However some work has been done in this area. This section discusses related research undertaken to improve the efficiency of schedulers through parallelisation. Frequently a Graphics Processor Unit (GPU) configuration has been used in this related work.

GPUs have been utilised to create massively parallel systems to improve computation in a variety of areas, for example in complex animation rendering, complex mathematical calculations and big data processing (Creel and Zubair 2012, Jung, Gnanasambandam, and Mukherjee 2012, Ponce et al. 2012, and Peng and Nie 2008).

Nesmachnow, Cancela and Alba (2011 and 2012) investigated the use of massively parallel GPUs (Graphical Processing Units) to improve scheduling time. In 2011 they implemented the MinMin and Suffrage algorithm on GPU architecture (Nesmachnow and Canabe 2011). They recorded improvements in scheduling time when the number of tasks goes beyond 8000 and where number of machines is more than 250. In their experiment the number of machines (GPUs) was 32 times less than the number of tasks. In 2012, the same researchers applied four variants of the parallelism on the MinMin algorithm and obtained large improvements in computation time when using parallel scheduling in comparison to serial

scheduling when the number of tasks increases. The proposed parallel method demonstrated a significant reduction on the computing times with the parallel GPU hardware (Canabe and Nesmachnow 2012).

Other researchers have proposed genetic and memetic algorithms which exploit GPUs in solving the scheduling problem. Nesmachnow and Mauro (2011) have presented CPU and GPU multi-threaded parallel designs of the MinMin algorithm. As would be expected, the GPU design outperforms the CPU because of the massive parallelisation. The parallel CPU solution outperformed the serial algorithm. Pinel, Dorronsoro and Bouvry (2013) proposed a cellular genetic algorithm (CGA) to minimize the batch scheduling of independent tasks. The work was more intended to reduce makespan than increase scheduling throughput. The CGA brought more accurate results than some previous algorithms but took longer to run. Mirsoleimani, Karami and Khunjush (2013) propose a memetic algorithm, which uses combinations of non-deterministic approaches to solve the scheduling problem in a GPU environment. Significant improvement in speedup was recorded.

The difference between the above related work and this research is that most related work which concentrates on parallelisation of the scheduler has focussed on a GPU environment and/or on non-deterministic algorithms such as genetic or memetic algorithms. The GPU environment offers massive parallelisation. However non-determinist algorithms have unpredictable run times. The scope of this research has been the more general purpose environment which was selected to avoid creating a facility that requires a specialised environment. The method also concentrated on deterministic algorithms to have better control on scheduler execution time. The Pinel, Dorronso and Bouvry (2013) work on CGA is related to this research except that their work focused on using a non-deterministic algorithm to reduce makespan rather than increasing scheduling throughput. Also the second variant of MinMin in Canabe and Nesmachnow (2012) is similar to this work except that the MCT of the jobs from the N domains partitions are computed separately by each GPU on all the machines. This research differs in the novel use of grouping of machines and jobs to achieve greater scheduling efficiency through parallelisation.

Parallel multi-scheduling can improve Grid scheduling performance and should be exploited. The aim is therefore to exploit the use of multicores both on the scheduler and on the Grid sites. This is achievable through the innovative grouping algorithm developed in this

research. The development of the Group-based Parallel Multi-scheduler (GPMS) is an attempt to steer Grid scheduling algorithms towards tapping into the benefits of multicore systems in order to enhance performance.

Whilst developments in Grid scheduling have produced some beneficial results, there remains a gap that needs to be addressed. The algorithms described in this section focused on the scheduling of parallel tasks and not on parallelisation of the actual scheduling task itself. They never explicitly exploited the underlying multicore hardware in their executions. With both multicores and Grid computing becoming increasingly pervasive, it would be promising to harness the advantages of multicore to improve scheduling on the Grid.

The prospect for the future growth of the Grid examined by Klusácek et al. (2008) and Robert (2012) coupled with the achievements made with parallelism on scheduling calls for a paradigm shift and a dynamic approach to scheduling Grid jobs. A parallel scheduler for the Grid will facilitate increased throughput and scalability. The challenge is to develop a scheduler that is dynamic, optimizes resource utilization and above all increases scheduling-throughput.

2.9 Group Scheduling and Load Balancing

Group scheduling is an important solution that is used in this research. Previously some researchers have introduced group scheduling but have used it in a different way. A fore-runner to group scheduling was Gang scheduling. This section discusses the notion of gang scheduling and then the notion of grouping of jobs. Then, it discusses the relationship of this research to group scheduling. Finally, it addresses loading of machines by discussing load balancing in the GPMS.

2.9.1 Gang Scheduling

Gang scheduling according to Papazachos and Karatza (2009) concerns jobs or tasks consisting of a number of interacting subtasks which are scheduled to run simultaneously on distinct processors. The notion of gang scheduling was first introduced as co-scheduling to enable the concurrent execution of different parts of cooperating processes on a multiprocessor system and equally to solve the problem of blocking and thrashing that was

occasioned by the inability of then operating systems to coordinate cooperating processes (Ousterhout 1982).

Zhou, Walsh and Brent (2000) proposed the idea of dynamically *repacking* subtasks to schedule gangs based on time-slots or time-sharing and also based on space-sharing to create room for incoming processes and eliminate the problem of fragmentation. They also introduced the use of *workload tree* to ease search process for empty slots. Karatza (1999) extended gang scheduling into distributed systems prone to processor failure. The study showed improvement in performance when the mean repair time is low. Wiseman and Feitelson (2003) proposed a means of slightly relaxing the strict allocation of processes to processors, and allowed pairs of jobs with alternate CPU requirements like I/O bound jobs and CPU bound jobs to be carefully mixed-and-matched and scheduled to execute on same processor to complement I/O cycles of one job with CPU cycle of its complementary match. The approach greatly improved system utilization. Zhang et al. (2000) and Papazachos and Karatza (2009) proposed a dynamical migration scheme to dynamically migrate some tasks from one node to another node or one queue to another queue during execution of job. Both approaches aim to create enough space in one node or queue for waiting jobs and filling up fragmented space in another node or queue.

These studies of *gang-scheduling* showed that useful results can be attained with proper coordination of tasks and processors. Though gang-scheduling deals with the careful selection of related tasks for allocation to processors, it leads to the possibility of *multi-scheduling in Grid* where several independent tasks are selected and dispatched in groups onto several Grid resources for processing.

2.9.2 Grouping of Jobs

Grouping of jobs to optimize scheduling in heterogeneous systems and Grid is a well researched topic; grouping of jobs has been employed to enhance job sharing, distribution among nodes and improve resource utilization. Braun et al. (1998) implemented a method to take advantage of heterogeneous computing systems by decomposing application tasks into subtasks where each subtask is computationally homogeneous. The algorithm also involves matching and scheduling groups of tasks to the heterogeneous machines.

Ernemann et al. (2002) presented the potential benefits in sharing jobs among independent sites in the Grid environment and discussed a method of parallel multi-site job execution. Their work proved that significant improvement can be achieved in response time and that the use of multi-site applications can improve the results even more if the communication overhead can be kept to a limiting value. Grouping of small (fine-grained) jobs to form bigger (coarse-grained) jobs before scheduling to resources was exploited to primarily reduce the overhead of communication computation ratio (CCR) that always negates the advantages of distributed computing. This method was employed by Buyya et al. (2004) and Muthuvelu et al. (2005) to pack or group jobs before transmitting to Grid resources for computation. When jobs are grouped before scheduling, the computation time is reduced by a factor and performance of the scheduling process is also improved. Muthuvelu et al. (2005) presented a scheduling strategy that performs dynamic job grouping activity at runtime. The method employed granularity size to determine the total number of jobs that can be processed within a specified time and uses that to dynamically assemble individual fine-grained jobs of an application into a group of jobs and send the group of (coarse-grained) jobs to the Grid resources.

Muthuvelu et al. (2005) implemented the Grouping-based job scheduling algorithm that groups the jobs according to MIPS of the resource. The method reduced the communication time and processing time of the job, but the algorithm did not take other resource characteristics into account. As a result, the method employed individual resources or processing elements rather than group resources. Also, the grouping strategy did not employ parallelism. Hence, resources where not utilized sufficiently.

Keat et al. (2006) proposed a scheduling framework for bandwidth-aware job-grouping-based scheduling in Grid computing. In a related work, Liu and Liao (2009) also implemented grouping based fine-grained job scheduling in Grid computing. The methods group the jobs based on MIPS and Bandwidth of the resource. The methods use a Greedy algorithm to cluster lightweight jobs. A job is not allowed to be grouped but immediately scheduled to a resource if the job is a coarse-grained job.

SCOJO (Sodan et al. 2006) employed priority method to classify jobs into groups. Priorities are assigned to jobs according to the job-runtime classes which are short, medium, and long.

Short jobs are allocated higher priorities while long jobs get the lowest priority. An aging scheme (priority increase over time) method is also implemented to prevent starvation.

He, Hsu and Leiserson (2007) employed the DEQ (Dynamic-Equipartitioning) job scheduler to dynamically partition jobs in order to give each job a fair share of processors. If a job cannot use its fair share, DEQ distributes the extra processors across the other jobs. Franke, Lepping and Schwiegelshohn (2007) used resource consumption as a criterion to group users' jobs which were categorised into five groups. Group 1 represents all users with high resource consumption, whereas group 5 represent users with very low resource consumption. The work concentrated on only parallel jobs and uses only identical machines for processing.

Selvi et al. (2010) proposed a rough set engine to group similar jobs and identifies the group to which the newly submitted job belongs. Soni et al. (2010) proposed the Constraint-Based Job and Resource Scheduling (CBJRS) algorithm. The method group jobs based on processing capability (in MIPS), bandwidth (in Mb/s), and memory-size (in Mb) of the available resources. The resources are arranged in hierarchical manner where Heap Sort Tree is used to obtain the highest computational power resource or root node, so as to make balanced and effective job scheduling.

Sharma et al. (2010) employed job grouping to maximize resource utilization, scalability, robustness, efficiency and load balancing ability of the Grid for scheduling of jobs in Grid computing. This method also targetted independent tasks just as does the GPMS.

Vishnu, Raksha and Manoj (2010) proposed the 'Grouping-Based Job Scheduling Model in Grid Computing', a model that explored the advantages of grouping *light-weight* or small jobs to *coarse-grain* jobs before scheduling to reduce the CCR. The method employed a First-Come-First-Serve method in the final schedule and does not consider parallelisation of the scheduling task.

2.9.3 Relationship of this Research to Previous Research in Grouping

The research described in 2.9.1 and 2.9.2 shows how it can be advantageous to group jobs. Gang scheduling (section 2.9.1) is different to the use of grouping in this research because it is concerned with grouping jobs that are inter-dependent in some way. This research focuses

on grouping independent jobs. The grouping used in previous research described in section 2.9.2 has similarities with this work.

Two general approaches emerged from the study of grouping methods described in section 2.9.2. The first is grouping of jobs from fine-grained to course-grained in order to reduce communication costs and the second is grouping of jobs or machines according to characteristics so that jobs can be assigned to the most suitable machines. This research utilises the latter approach in that similarities of machines may be taken into account as a possible grouping method but the focus of this research is not on the scheduling of jobs to machines per se but on the parallisation of the actual scheduling activity, the parallisation of the scheduler. This was not address in the related work on grouping that has been described.

The intention is that grouping of jobs will allow the jobs in each group to be treated as a scheduling entity accessible by discrete threads. Job grouping will therefore be explored in this investigation to enable jobs to be multi-scheduled. The researcher calls this method the Group-based Parallel Multi-Scheduling (GPMS) method.

For jobs to be adequately scheduled in parallel such jobs must first be collected in an order for them to be scheduled at same time, hence grouping of a set of tasks or jobs is necessary before multi-scheduling. This research focuses on group scheduling and refers to the election or selection of a (several) independent tasks from a job group and dispatching or scheduling to machine groups.

Group–based multi-scheduling (Abraham, James and Yaacob 2015a, and Abraham, James and Yaacob 2015b) aims at splitting Grid jobs and machines into the same number of groups. Machine groups are then paired with job groups, and then independent threads are utilized to execute scheduling algorithms within the groups and between paired groups. Using this method, there is a high guarantee that if jobs are equitably distributed into the groups, and the threads are executing independently unhindered in separate cores, and the jobs are highly independent, then the method can improve scheduling efficiency by large margins. Multi-scheduling allows multiple independent scheduling instances to occur simultaneously within the groups. This will enable the parallelisation based on threads, allowing them to independently access the groups to schedule jobs based on the scheduling policy.

The method uses each thread to execute the scheduling algorithm independently in a group, automatically improving the total scheduling time of the grouping method by factors approaching or greater than N, if N is the number of groups used.

2.9.4 Load Balancing

Dynamic load balancing have been extensively studied in distributed systems (Dos Santos 1996). The GPMS proposed in this research exploits load balancing. In this section previous work in load balancing and its relationship to this research is considered.

Shivaratri, Krueger and Singhal (1992) studied load-balancing algorithms in heterogeneous Networks. Roussopoulos and Baker (2006) studied load-balancing issues in P2P context. Randomised load-balancing algorithms were popularised by work-stealing algorithms (Blumofe and Leiserson 1999, and Berenbrink, Friedetzky, and Goldberg 2003).

Cao et al. (2005) uses an algorithm based on an evolutionary process that uses intelligent agents and a multi-agent to deal with load balancing issues on the Grid. The method uses agents to schedule resource and to balance load across multiple host processors in a local Grid. An agent is capable of coupling application performance data with iterative heuristic algorithms to minimise makespan, host idle time and meet the deadline requirements for each task. The method can respond to system changes such as the addition or deletion of tasks, or changes in the number of hosts / processors available in a local Grid. Ungurean (2015) proposed an algorithm for scheduling and dynamic load balancing**.** The algorithm carries out scheduling of jobs towards nodes and the dynamic adjustment of nodes loaded into the system by transferring the jobs from loaded nodes towards the other nodes using the round robin algorithm. Algorithms to dynamically migrate jobs from highly loaded nodes to weakly loaded nodes are also implemented to enhance load balancing. The process of dynamically migrating jobs introduces some overheads with the algorithm.

These algorithms employ load balancing to distribute tasks to nodes and ensure that all processors are equally optimized. The GPMS system employs a grouping method also to ensure equitable distribution of jobs to Grid nodes in order to enhance scheduling throughput.

The GPMS proposed in this research exploits load balancing. It exploits grouping methods that balances jobs and machines into groups. For instance, the ETSB (execution time sorted and balanced) method employed ensures that jobs are evenly distributed across groups based on their size. Also, the EvenlyDistributed (EvenDist) method ensures that machines are evenly distributed into groups.

2.10 Summary

The chapter discussed relevant literature related to this thesis. It started with a general exposition of the Grid. Then it discussed parallelism, focussing on technological developments yielding pervasive multicore systems, constraints of multicores and the need for their greater exploitation through parallelism. The relationship of the Grid to parallelism was then explored. Next previous research in distributed and high throughput computing was elucidated as well as earlier endeavours in the design of parallel scheduling algorithms, including nature-inspired algorithms and its inherent parallelism features. The chapter went on to focus on Grid scheduling algorithms, discussing both deterministic and nature-inspired methods. Research in gang scheduling, group scheduling and load balancing was finally explored in relation to the new method proposed in this research.

An observation was made that insufficient attention has been paid to the parallelisation of the Grid scheduling task.

The next chater discusses the methodology: the stages employed in achieving the aims and objectives and as discusses the motivation for applying the method.

CHAPTER THREE

RESEARCH QUESTION AND METHODOLOGY

CHAPTER THREE
RESEARCH QUESTION AND METHODOLOGY

3.1 Introduction

It has been noted that multicore systems are on the increase, yet Grid scheduling algorithms do not typically exploit the opportunities of parallelism on multicores for the actual scheduling task. Current Grid schedulers are sequential and thus can get overwhelmed with increased workload thereby creating bottlenecks in Grid scheduling.

This research proposes a job and machine grouping methods which are aimed at enhancing Grid scheduling by generating several independent scheduling instances between independent groups of jobs and groups of machines in parallel. In the following sections, the method used to develop an appropriate multi-scheduling approach is described.

3.2 The Identified Gap

Hardware computing technology has shifted grounds and multicore computers are now on the increase. While these advances hold much promise for the future of computing, the same case cannot be argued for sequential algorithms. It is contended that the benefits of the multicore technology should be exploited by engineering more applications which adopt parallelism. Current Grid scheduling algorithms do not exploit parallelism. Hence, the lack of a dynamic method to meet the future needs of Grid scheduling is the motivation for this research and the identified gap that will be addressed by this research. This research therefore aims to exploit parallelism to harness the benefits of multicores in Grid scheduling.

In the light of the above, the research question is:

How can multi-scheduling and parallelism be exploited to take advantage of multicores in order to improve the Grid job scheduling task?

3.3 Overview of Method

The following overall method was adopted in this research:

3.3.1 Literature Review

This stage involved a rigorous search and review of relevant and related literature. This was done to gain more knowledge in various fields relating to the research. Related literature was gathered and analysed in the following relevant areas: Grid; Parallelism; Distibuted and High Throughput Systems; Parallel Scheduling; Grid Scheduling Algorithms; and Group Scheduling.

3.3.2 Definition of Terms

After reviewing the literature and gaining more understanding of the Grid, multicores and current Grid scheduling algorithms, the keywords relating to the literature were then defined. These are listed in the glossary.

3.3.3 Research Question Development

The literature search opened up a gap, which was the dearth of Grid scheduling algorithms that take advantage of multicore computers to scale Grid scheduling to meet the future, based on the backdrop that multicore computers are already pervasive and making their way into every aspect of our computing lives. The research question was generated to reveal the motivation for the research, expose the identified gaps in current systems and provide the direction for the design of a solution.

3.3.4 Solution Design and Development

After the identification of gaps in the area of research, and the subsequent generation of the research question, the solution for the research was developed. This stage involved first the design of a Grid scheduling model that describes the components and their functionalities. Different design aids were used. These included:

- Flowchart was used to visualize some operations of the system.
- Pseudo codes and algorithms were used to describe the logical operations of the system.
- Context diagram was used to represent the system and sub-systems of the GPMS.

- UML diagrams were used to describe operations in the system. The UML diagrams used include:
  - Use Case diagram: was used to describe the interaction between the user and the system
  - Activity diagram: was used to describe the various activities within the system
  - Sequence diagram: was used to describe the sequence of operations within the system
  - Class diagram: was used to show the functional classes, methods and attributes of the system.

The solution was developed in two stages, both of which were based on the idea of groups of jobs, groups of machines and simultaneous instances of scheduling. First the Priority-based Parallel Multi-Scheduler (PPMS) was designed and developed. The PPMS used the Priority method of job grouping which is described in section 3.3.4.1and also in more detail in Chapter Four. After observing some weaknesses in the PPMS method, the Group-Based Parallel Multi-Scheduler (GPMS) was designed and developed and is described in section 3.3.4.2 and also in more detail in Chapter Four. The GPMS is a more generic solution which can incorporate multiple methods of grouping including the Priority method which was the underlying method for the PPMS. On the other hand, the PPMS was structured such that it could only support the Priority method. Later the GPMS was generalised to incorporate the Priory method as an additional method. The programming language used for the development of the scheduler was Java.

3.3.4.1 The Priority-based Parallel Multi-scheduler (PPMS)

The Priority-based Parallel Multi-scheduler (Goodhead, James and Yaacob 2014) exploits parallelism on multicores both at the scheduler and at the Grid resources level. Jobs were split or categorized into four priority groups based on their attributes. Machines (Grid resources) were also distributed into four groups based on their configuration. Job groups were then paired to machine groups and the scheduling algorithm was executed independently within paired job-machine groups. Parallelism was implemented using independent threads within job-machine pairs. The researcher named the method of grouping used in the PPMS, the Priority method.

3.3.4.2 Group-based Parallel Multi-scheduler (GPMS)

The Group-based Parallel Multi-scheduler (Goodhead, James and Yaacob 2015) aimed at exploring further the advantages of grouping jobs and machines and multi-scheduling in parallel on multicore systems to enhance scheduling algorithms in Grid. The Priority method of job grouping used in the PPMS had been found to be handicapped in that the effects of the priority grouping could not be ascertained since the number of groups was constant. Secondly, most jobs in the experimental source file had similar priority, hence were allocated to a single group. This affected the overall performance. The GPMS method was therefore developed to remedy the inadequacies of the PPMS.

The GPMS employed two methods in grouping jobs, these were:

Execution Time Balanced (ETB)–this method estimates the execution of all the jobs then distributes them equally (balanced) among groups.

Execution Time Sorted and Balanced (ETSB)–this method estimates execution time of all jobs, then sort jobs and then distributes them equally (balanced) across groups.

3.3.4.3 Machine Grouping Method

A machine group contains a set of different computers or Grid resources for servicing a set of jobs from a job group. Information about Grid machines i.e. MachineId, CPUSpeed and number of CPUs are used for the grouping of machines and also for simulation and computation of execution times of jobs. The same machine grouping method was used in both PPMS and GPMS.

Machines are split into groups based on their configurations; two methods adopted to categorize machines into groups are:

Similar Together (SimTog) - This method takes machines with similar characteristics or configuration into the same group. Similarity of configuration is based on the speed of the processor and numbers of CPUs. Due to the fact that the machines are not equally spread based on configuration, this method may generate some groups with more powerful machine configuration than other groups. Groups with better machine configuration may perform jobs quicker than those with less powerful configurations. Under these circumstances, if the same

numbers of jobs are scheduled to all groups, the group with less powerful machines may get busier and extend the overall completion time of scheduling and of job execution. On the other hand, QoS could be served well in this method by allocating high priority jobs to the most powerful machine group.

Evenly Distributed (EvenDist)-This method distributes machines with different configurations equally into all the groups; it ensures that all groups get equal share of the various machine configurations. This method should favour algorithms whose policies does not favour any particular set of jobs.

3.3.5 Simulation

This phase involved the simulation of an environment to test the effectiveness of the systems. Simulation was done due to the difficulty in accessing real Grid infrastructures. Both the Grid environment and the execution of jobs on machines were simulated.

Simulation of Grid

The Grid environment was simulated as comprising of four Grid sites, each Grid site contained different configurations of machines, and the machines further composed of various numbers of CPUs.

Simulation of Job Execution and Completion Time

The execution of jobs on machines and the completion time of jobs were simulated based on the size of the job and the speed of the machine it was assigned to with reference to a standard machine which in this case is a machine with 1GHz and 1 core or CPU.

3.3.6 Experimentation

Experimentation was carried out in phases. Seven different experiments were carried out. In the first instance, the MinMin scheduling algorithm was executed to schedule a range of jobs. This first experiment is treated as the base experiment and results from this experiment were compared against results from the other experiments.

The second and third experiments used the Priority job grouping method in combination with the two machine grouping methods (i.e. SimTog and EvenDist). At the time of this experimentation, the scheduler used was the PPMS. The fourth and fifth experiments used the **ETB** method in combination with the two machine grouping methods. And lastly, the sixth and seventh experiments used the **ETSB** method in combination with the two machine grouping methods. At the time of this experimentation, the scheduler used was the more generic GPMS. The experiments were executed on one of Coventry University's HPC systems known locally as Pluto.

The experiments used the simulated Grid environment with four Grid sites consisting of machines with different CPU speeds and numbers of processors. Machines from the various Grid sites were grouped based on their configurations. In the scheduling stage, jobs were dispatched directly to the CPUs on the individual machines.

Three job grouping methods (Priority, *ETB* and *ETSB)* and two machine grouping methods (*EvenDist* and *SimTog*) were used in the experimentation. Except for the case of the Priority method which used four constant groups, the number of groups was varied between 2, 4, 8 and 16 and the number of threads as varied from 1 to 16 (in steps of power 2). The MinMin Grid scheduling algorithm was then executed independently within the groups.

The MinMin algorithm (referred to as the ordinary MinMin) was first executed to schedule a range of tasks and the time recorded appropriately. Then, the various grouping methods were executed to group jobs and machines. Thereafter, the MinMin algorithm was executed again to schedule same range of jobs independently within the groups and the time taken to complete each range of jobs using the various grouping methods and also varying the number of groups and threads was recorded. The various results were later compared to the ordinary MinMin and to each other. The MinMin algorithm was chosen for the comparative analysis because most research in Grid scheduling also compares with the MinMin and one can say that researchers have almost turned the MinMin algorithm into a base algorithm for comparison in Grid scheduling algorithms research.

Table 2 summarises the experiments carried out. For each experiment, job sets ranging in number, from 1000 to 10000 in steps of 1000 were used and also a range of threads were used, 1,2,4,8, and 16. For each variation in each experiment the scheduling time and

makespan was recorded. In all, measurements for 950 experimentation variations were recorded for analysis. Table 3 shows the variations for each experiment.

Table 2 Scheduling Experiments

Experiment Number	Number of Groups	Scheduling Acronym	Job Grouping Method	Machine Grouping Method	Inside Group Scheduling Method
1	1	Ordinary MinMin	n/a	n/a	MinMin
2	4	Priority-SimTog	Priority	SimTog	MinMin
3	4	Priority-EvenDist	Priority	EvenDist	MinMin
4	2,4,8,16	ETB-SimTog	ETB	SimTog	MinMin
5	2,4,8,16	ETB-EvenDist	ETB	EvenDist	MinMin
6	2,4,8,16	ETSB-SimTog	ETSB	SimTog	MinMin
7	2,4,8,16	ETB-EvenDist	ETSB	EvenDist	MinMin

Table 3 Number of Variations of each Experiment

Experiment Number	Scheduling Acronym	Number of Grouping Variations	Number of Job Sets (Input job set size variations)	Number of Threads Variations	Number of Experiment Variations
1	Ordinary MinMin	1	10 (1000 -10000 in steps of 1000)	5 (1,2,4,8,16)	50
2	Priority-SimTog	1 (always 4 groups)	10 (1000 -10000 in steps of 1000)	5 (1,2,4,8,16)	50
3	Priority-EvenDist	1 (always 4 groups)	10 (1000 -10000 in steps of 1000)	5 (1,2,4,8,16)	50
4	ETB-SimTog	4 (2,4,8,16)	10 (1000 -10000 in steps of 1000)	5 (1,2,4,8,16)	200
5	ETB-EvenDist	4 (2,4,8,16)	10 (1000 -10000 in steps of 1000)	5 (1,2,4,8,16)	200
6	ETSB-SimTog	4 (2,4,8,16)	10 (1000 -10000 in steps of 1000)	5 (1,2,4,8,16)	200
7	ETSB-EvenDist	(2,4,8,16)	(1000 -10000 in steps of 1000)	5 (1,2,4,8,16)	200

3.3.7 Analysis of Results

This stage involved the application of analysis tools on the results to derive meaning from them. The results from the experiments were written to an output file in text format. The results were then imported into a statistical analysis tool for analysis. First, the results were categorized based on methods, groups and threads before further analysis was performed.

3.3.7.1 Statistical analysis

Various statistical measures were used for the analysis of the data. The data manipulation and statistical analysis performed on the results included:

Sorting In order to process the vast amount of data produced by the experiments, custom level sorting was applied to different fields. This enabled the results to be sorted based on fields such as job grouping method, machine grouping method, number of threads and number of groups. This further enabled the application of mathematical formulas on each category of the result.

Total: After categorizing sets of result by methods, groups and threads, the total scheduling times for each set of result were computed for use in further analysis.

Mean: The means for each set category of the data were also computed.

Standard deviation: The standard deviation was computed between results from the different methods to show how the mean of the methods vary and also to reveal how one method performs better than the other.

Correlation: The correlation analysis was carried out to show the strength in relationships or randomness between the results of the different methods.

Analysis of variance: The analysis of variance was also performed between the result sets. This was done to show if there were significant differences between the result sets from the methods.

3.3.7.2 Mathematical formulas

Mathematical formulas and functions were also used to compute values used for further analysis and evaluation. The mathematical formulas and functions used were speed and improvement.

Speedup and improvement

The speedup was computed to evaluate the differences and gains made between the methods and the ordinary MinMin. The **speedup** was computed for each step (or interval of scheduling). While the improvement was computed to know the overall gain made over the ordinary MinMin by the methods, the **improvement** was computed using the computed overall **total** and or **average.** Both values were computed in multiples and in percentages with different formulae.

Speedup in multiple(X)

Speedup is the gains made when applying a parallelised algorithm compared to a serial algorithm to execute the same job. The speedup **in multiple** is obtained by method dividing scheduling time results of the MinMin algorithm (referred here as ordinary MinMin and implemented in the base experiment without grouping) by that of the applied grouping method. This value tells by how many times the applied method performs better than ordinary MinMin (the base method) at each step of the schedule.

Speedup in multiple was computed as:

$$\frac{MinMin_{schedtime}}{Group_{schedtime}}$$

Equation 1 Speedup (X)

Speedup in percentage (%)

The speedup in Percentage method uses the values of both the ordinary MinMin and the applied method to compute the percentage. This evaluation tells by how many percent the method performs better than the ordinary MinMin method. This value is obtained by subtracting the scheduling time of the group method at each stage from the scheduling time of the ordinary MinMin at the corresponding stage, then dividing by the scheduling time of the ordinary MinMin and multiplied by 100. The value is computed as:

$$\left(\frac{MinMin_{schedtime} - Group_{schedtime}}{MinMin_{schedtime}}\right) * 100$$

Equation 2 Speedup (%)

Performance Improvement over MinMin

The performance improvement is the overall gain made over a serial algorithm by the parallel algorithm. This value was computed in multiples and in percentage.

Performance Improvement in multiple(X)

The performance improvement in multiple was computed similar to the speedup in multiple but here, it is the cumulated total scheduling time for the method that is used. This value is computed with the total or average scheduling time of the ordinary MinMin divided by the total scheduling time of the GPMS method. The value is computed as:

$$\frac{Total_{MinMin}}{Total_{Group}}$$

Equation 3 Improvement over MinMin (X)

Performance Improvement in percentage (%)

This value represents the improvement over the MinMin algorithm in percentage. It is computed by subtracting the total scheduling time of the method from the total scheduling time of the ordinary MinMin then dividing by the total scheduling time of the MinMin and multiplying by 100. The performance improvement in percentage of the grouping method over the non-grouping method is computed as:

$$\frac{x_1 - x_2}{x_1} X100$$

where x_1 = MinMin Total Scheduling Time

x_2 =Group Methods Total Scheduling time

or $$\frac{MinMin_{TotalSchedTime} - Group_{TotalSchedTime}}{MinMin_{TotalSchedTime}} * 100$$

Equation 4 Improvement over MinMin (%)

Performance Improvement between successive groups

This computation is used to evaluate the improvement between two successive groups when using the same method. It is computed from the total scheduling times of a group and the total scheduling time of its successor group.

Performance Improvement between groups in multiple (X)

The performance improvement between two groups in multiple (x) is computed with the total scheduling time of the two groups. The value is computed by dividing the total scheduling time of the group by the total scheduling time of the successor group. For instance, the improvement between 4 groups and 2 groups or between 8 groups and 4 groups were computed as follows:

$$\frac{nGroup_{TotalSchedTime}}{n+1Group_{TotalSchedTime}}$$
$$n \in [2,4,8]$$

If nGrps $_{= 2\ Grps}$

Then n+1 Grps $_{=4Grps}$

Or If nGrps $_{= 4Grps}$

Then n+1 Grps $_{=8Grps}$

Equation 5 Improvement between groups (X)

Performance Improvement between successive groups in percentage (%)

This computation is used to evaluate the performance improvement between successive group cardinalities in percentage. It subtracts the total scheduling time of the successor group from the total scheduling time of the group, then divides by the total scheduling time of the group and multiplying by 100. The value is computed with the formula:

$$\frac{nGroup_{TotalSchedTime} - n+1Group_{TotalSchedTime}}{nGroup_{TotalSchedTime}} * 100$$
$$n \in [2,4,8]$$

Equation 6 Improvement between groups (%)

3.3.8 Evaluation of Results

This stage drew meaning from the analysed results and provided explanation to the analysis. The evaluation was carried out to ascertain the efficacy of the method and to appraise the overall success of the research. The outcome of this phase was used to ascertain if the research aim had been achieved and if the research question had been answered.

The evaluation was carried out against the MinMin algorithm. The motivation to compare against the MinMin algorithm was based on the fact that several other researchers (in Grid scheduling) also compared their results against the MinMin and the MinMin algorithm has almost become the coin of researcher's evaluation in Grid scheduling. The motivation for using MinMin is further justified in section 3.3.9.

Some of the results were plotted on charts to get clearer meaning. Methods used in the evaluation include:

***Charts*:** Different types of charts were employed to represent the data and analyse results graphically. Some charts used in the evaluation are line charts, pie charts, bar charts and column charts. From the charts, the differences in performance between the grouping *methods and the MinMin algorithm were presented graphically and were easy to deduce.*

*C**orrelation*:** Correlation test was carried out to determine the relationship or randomness between the results. The result of this test enabled more meaning to be made out of the *values.*

***ANOVA*:** Analysis of variance was performed between the means of the results. This was done to show if there were significant differences between the means of the methods. Results from this analysis helped informed the conclusion of this research

***T-test of significance*:** T-test was used to compare the significance differences between the mean of more than two variables.

***Standard deviation*:** The standard deviation of the different methods were also computed and contributed to the discussions of the research.

3.3.9 Motivation for using MinMin for Comparison

The MinMin heuristic is a simple deterministic algorithm initially proposed by Ibarra and Kim in 1977 for the problem of scheduling independent tasks. The MinMin algorithm starts with a set *U* of all unmapped tasks, and then, the set of minimum completion times *M* for each task in *U (*on each machine) is calculated. Then, the task with the overall minimum completion time (MCT) from *M* is selected and assigned to the corresponding machine (hence the name MinMin). Lastly, the newly mapped task is removed from the set *U*, and the process repeats until all tasks are mapped (i.e. *U* is empty). After each assignment, the availability status of the machines is updated.

The objective is to minimise makespan by assigning more tasks to the machines that can complete them the earliest and also execute them the fastest. The problem has primarily been evaluated in a static "off-line" context - where all tasks are known before scheduling begins, and the objective is the minimization of makespan, i.e. the time to finish all tasks. The algorithms can also be applied in the dynamic "on-line" context, by "unscheduling" all non-started jobs at each scheduling event - when either a new job arrives or a job completes.

The accuracy and simplicity of the algorithm has lend credence to it been used widely as reference in many research papers (Casanova et al.1999, Braun et al. 2001, Sabin et al. 2003, Ritchie and Levine 2004, Dong and Akl, 2006, Luo, Lu and Shi 2007, Hao, Liu and Wen 2012, and Prajapati and Shah 2014).

MinMin can be easily adapted to different scenarios. Hence, it has been adapted for the design of other efficient algorithms. For example, He, Sun and Laszewski (2003) propose a QoS Guided MinMin heuristic which guarantees the QoS requirements of particular tasks and minimizes the makespan at the same time. Wu, Shu and Zhang (2000) proposed a Segmented MinMin algorithm, in which tasks are first ordered by the expected completion time (it could be the maximum ECT, minimum ECT or average ECT on all of the resources), then the

ordered sequence is segmented, and finally MinMin is applied to all the segments. Other works proposed to improve the MinMin are (Dorronsoro et al. 2010, Xhafa et al. 2008a, and Xhafa et al. 2008b).

MinMin has also been proposed for scheduling tasks on heterogeneous systems (Freund et al. 1996, Freund et al. 1998, Maheswaran et al. 1999, Venugopal and Buyya 2008, Parsa and Entezari-Maleki 2009, Hephzibah and Easwarakumar 2010, Nesmachnow and Canabe 2011, and Nesmachnow, Cancela and Alba 2012).

According to Nesmachnow and Canabe (2011), the computational complexity of MinMin heuristics is O (N3), where N is the number of tasks to schedule. When solving large instances of the heterogeneous computing scheduling problem (HCSP), large execution times are required to perform the task-to-machine assignment (i.e. several minutes for a problem instance with 10000 tasks). This informed their decision to parallelise the MinMin in order to reduce the execution times required to find the schedules. Nesmachnow and Canabe 2011 proposed methods of parallelising the MinMin scheduling algorithm for GPUs and compared results against the serial version implementation. Also, Pinel, Dorronsoro and Bouvry (2013) presented a parallel version of MinMin in their work on cellular genetic algorithm (CGA) to minimize the batch scheduling of independent tasks.

Ye, Rao and Li (2006) noted that the MinMin algorithm is becoming the benchmark of resources scheduling problems in Grid. Hence, our decision to use the MinMin as basis for comparison was informed by the fact that the MinMin scheduling algorithm has been used extensively in previous studies on Grid scheduling and in parallel scheduling. Appendix D shows some research work that used MinMin for comparison.

3.4 Summary

This chapter started by summarising the problem area and stating the research question. It then described the methods used in the research which had led to the formulation of the research question and which subsequently served to address the research question. Methods used at each stage of the research were explained. The overriding methods used in this research have been prototype design, simulation and experimentation leading to an answer to the research question and a better understanding of grouping methods in Grid scheduling. The chapter also discussed the motivation for using the MinMin algorithm as basis for comparison.

The next chapter shall discuss the design of the GPMS.

CHAPTER FOUR

DESIGN OF THE GROUPING BASED MULTI-SCHEDULER

CHAPTER FOUR
DESIGN OF THE GROUPING BASED MULTI-SCHEDULER

4.1 Introduction

This chapter describes the design of the Group-based Parallel Multi-scheduler (GPMS), the simulations used and experimental test bed. The initial design focused on the Priority grouping method as that was the first concept, but the group-based method would later be made into a more general model due to the shortcomings that were observed with the Priority-based method.

This design process followed the strict adherence of laid down principles in software design and involved the use of design tools and graphical languages such as flowcharts and UML diagrams for visualizing, specifying, constructing, documenting and refining the functions and components of the system.

Generally, design tools are used for visualising, specifying and documenting the components of a software intensive system as they provide basis for modelling use cases and scenarios to define and refine functionality that a software system is expected to provide. UML was chosen among other design tools because it provides strongly defined meaning and clarity for every element and encourages understanding of the task.

4.2 Design of the Group-based Parallel Multi-Scheduler

The Group-based Parallel Multi-scheduler (GPMS) was designed to take advantage of the underlying hardware of multicore systems. The GPMS is intended to execute on a multicore system. Furthermore, the machines making up the Grid (Grid resources) are assumed to be composed of different numbers of multicores. In the following sections, the functions and components of the GPMS are described.

4.2.1 Functions of the Group-based Parallel Multi-scheduler

The functions of the system are outlined in the Table 4.

Table 4 Functions of the GPMS

Functions of the Grouping-Based Parallel Multi-scheduler for Grid
i. Employ grouping methods for both jobs and machines.
ii. Split users jobs into groups based on job attributes
iii. Split Grid machines also into groups based on some criteria
iv. Pair groups of jobs to groups of machines
v. Schedule jobs to machines between paired groups in parallel (Multi-scheduling) – the scheduling of jobs targets the cores of the Grid machines
vi. Time the scheduling and processing activities

After specifying the overall functions of the system, the requirements of the system which determine the components to perform such functions were formulated. This was done with the 'shall statement' used to state what the multi-scheduler will do.

4.2.2 The 'Shall Statement' and System Requirement

The 'Shall Statement' describes in a nutshell the functionality the system shall provide and by extension the components required that can enable the system to perform such functions. The GPMS **shall:**

- Take as input a batch of jobs into the system – this requires a means of reading data into the system.
- Group jobs based on their attributes – this requires a means of identifying jobs attributes and a means of categorizing the attributes.
- Take as input a set of Grid machines. Hence a means of identifying and registering Grid machines is required.
- Categorize Grid machines (Grid resources) based on configurations. Hence a means of comparing and identifying machine configurations shall be required.

- Schedule jobs between machine groups and job groups. Hence a means of multi-scheduling jobs independently from the groups is required.
- Dispatch scheduled jobs to Grid machine cores.
- Know the number of jobs scheduled at every instance. Hence a means for counting is required.
- Be able to know the status of jobs being executed.
- Receive executed jobs from the Grid sites
- Returns jobs to users
- Monitors scheduling activities.

The functional components for the GPMS were deduced and are shown in Table 5.

Table 5 Functions of the GPMS components

S/No	Functions
1	A component that provides a means to accept users jobs into the system
2	A component that determines job attributes in order to group jobs
3	A component that determines machine attributes and group them
4	A component that schedules jobs to machines
5	A component that dispatches jobs to Grid resource
6	A component that receives executed jobs from Grid resource
7	A component that returns results to users
8	A component that monitors scheduling activities like counting and timing

4.2.3 Context Diagram

The context diagram views the system as a black box. Details of the internal operations are not seen. It indicates how external events affect the system and how internal/system events affect the outside world. Figure 4 is made up of two context diagrams 4a and 4b. Figure 4a shows the GPMS system as a black-box and Figure 4b shows the sub-systems or units within the GPMS.

The JobReader is the component that accepts jobs into the system; the JobGrouper determines jobs attributes and also categorizes or groups jobs. The MachineGrouper determines machines configuration and group machines. The Multischeduler is the component that enables the parallel execution of the scheduling algorithm within paired groups while the JobDespatcher is the component that despatches jobs to machines at Grid sites. The JobReceiver receives processed jobs from the Grid sites and returns them to users.

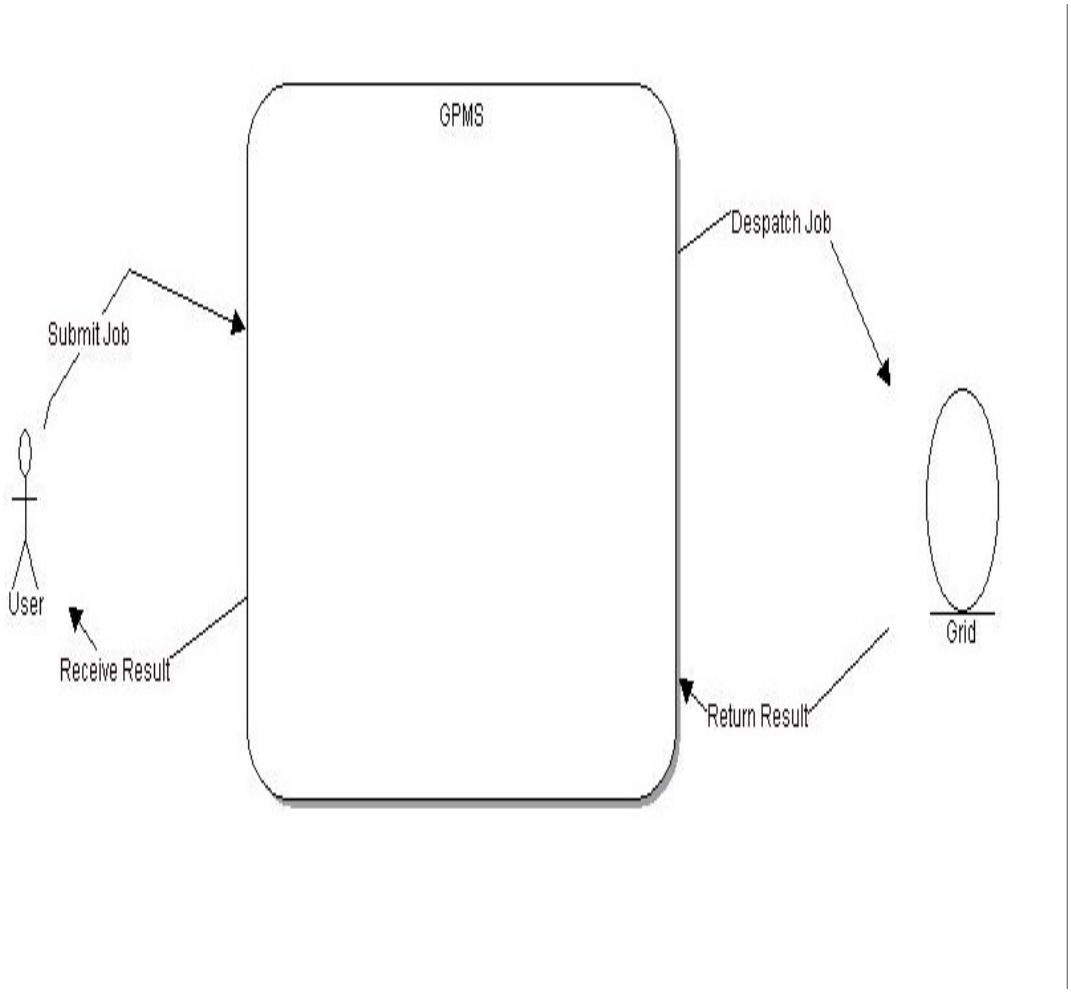


Figure 4a: The GPMS as a black box

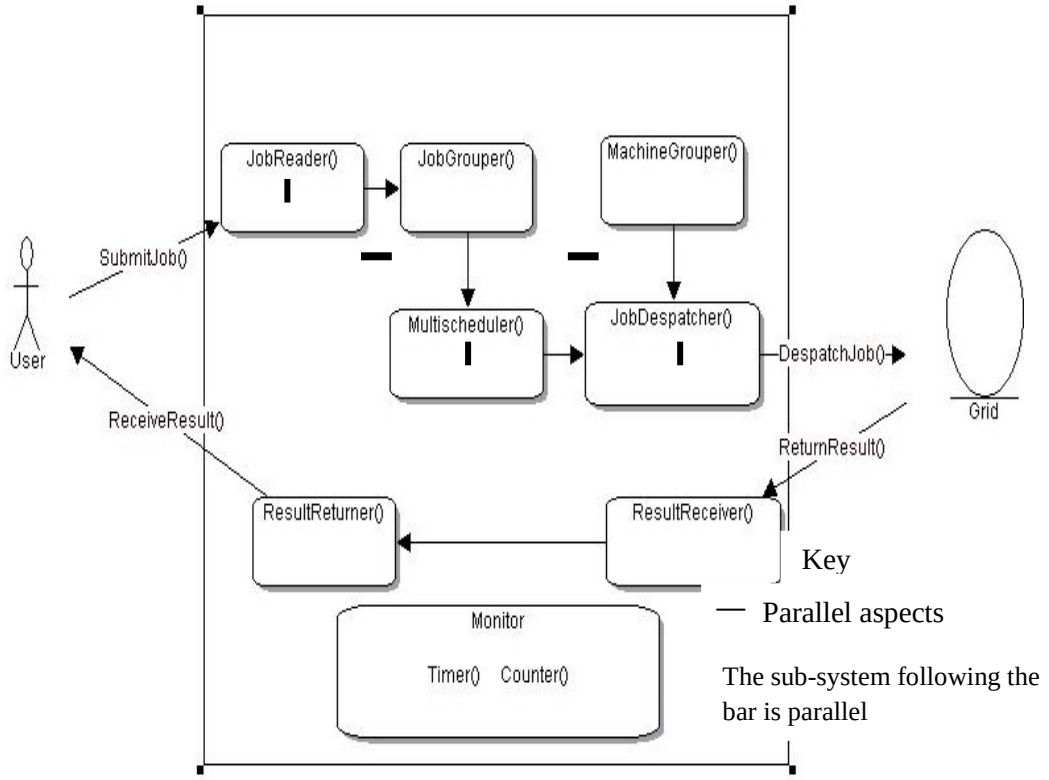


Figure 4b: Sub-systems within the GPMS

Figure 4a and 4b: Two level Context diagram for the GPMS system

The interface between the outside world (users) and the system is the job input screen. Beyond the interface is the system itself. The system contains subsystems which carry out specific functions like accepting users jobs (input reader), sorting/grouping (job sorter), categorizing machines (machine sorter), scheduling jobs (multi-scheduler) and the dispatcher. These are represented as a black box at the subsystem level. The subsystems for the GPMS comprise the major components that enable the system to function as a whole. These are:

Job Reader: Reads job input from file and feed to the system

JobGrouper: sorts the jobs into groups based on job attributes

MachineGrouper: categorizes machines into groups based on their configuration

Multi-scheduler: schedules jobs from different job groups to machine groups.

Job Dispatcher: dispatches jobs to machines at Grid sites

Job Receiver: receives jobs from the Grid

Job Returner: returns results to the user

Monitor: records the scheduling activities like timing and counting

4.2.4 Use Case Diagram

A Use Case diagram depicts how a system is intended to be used; it shows the intended functionality of the system and how users will use it. Figure 5 shows the Use Case of the GPMS systems and some high-level view of the functionality of the system.

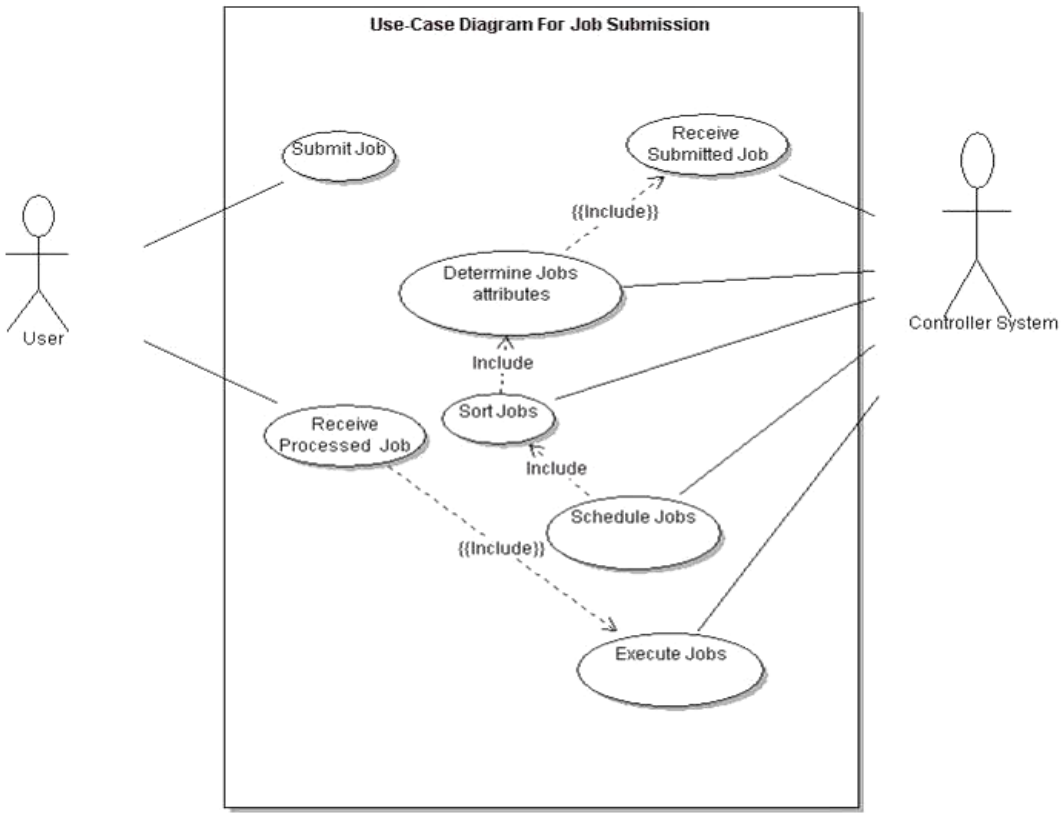


Figure 5: Use Case diagram for the GPMS system

4.2.5 Activity Diagram

The activity diagram shows the procedural flow of activities associated with the system. The activity within the system starts with the system polling for the availability of jobs. If there

are no jobs the system continues polling. Once jobs are available, the JobReader reads the jobs into the system, the JobGrouper then determines jobs attributes and based on the attributes, it groups (sort jobs into groups). If there are Grid machines, the MachineGrouper determines machines configuration and groups machines based on the configuration. In this consideration, it is assumed that Grid machines are always available; this is because scheduling on the Grid differs from traditional scheduling in other environments. Jobs or processes compete for limited resources in most traditional environments, whereas on the Grid, computing machines are always available at different locations. The scheduling constraint therefore is not mostly how to ration scarce resources for competing processes but how to meet certain user requirements. After grouping both jobs and machines, the Multi-scheduler then pairs job groups and machine groups and executes the 'selected' scheduling algorithm within paired groups. Jobs from within the groups are then scheduled to machines within machine groups. The JobDespatcher despatches the jobs to machines at Grid sites. After the execution, results are returned to users and the system continues the cycle.The JobReceiver receives processed jobs from the Grid sites and the ResultReturner returns the result to users. It will however be noted that the system does not have to wait for jobs to be completed before starting the next round. At every point jobs become available; the system is activated and begins the processes (see Figure 6). However, machines are more stable on the Grid than jobs. Sorting machines as frequently as jobs at every scheduling operation requires extra overhead. As a result, sorting of machines was carried out less frequently (in the experiment). Machines were sorted only when the grouping method changes or when the number of groups to be used changes. Moreover, if the number of machines does not change, then the system has a way of remembering a previously used machine list. Hence, in real life systems, the grouping of machines can be less frequent than in the experiment.

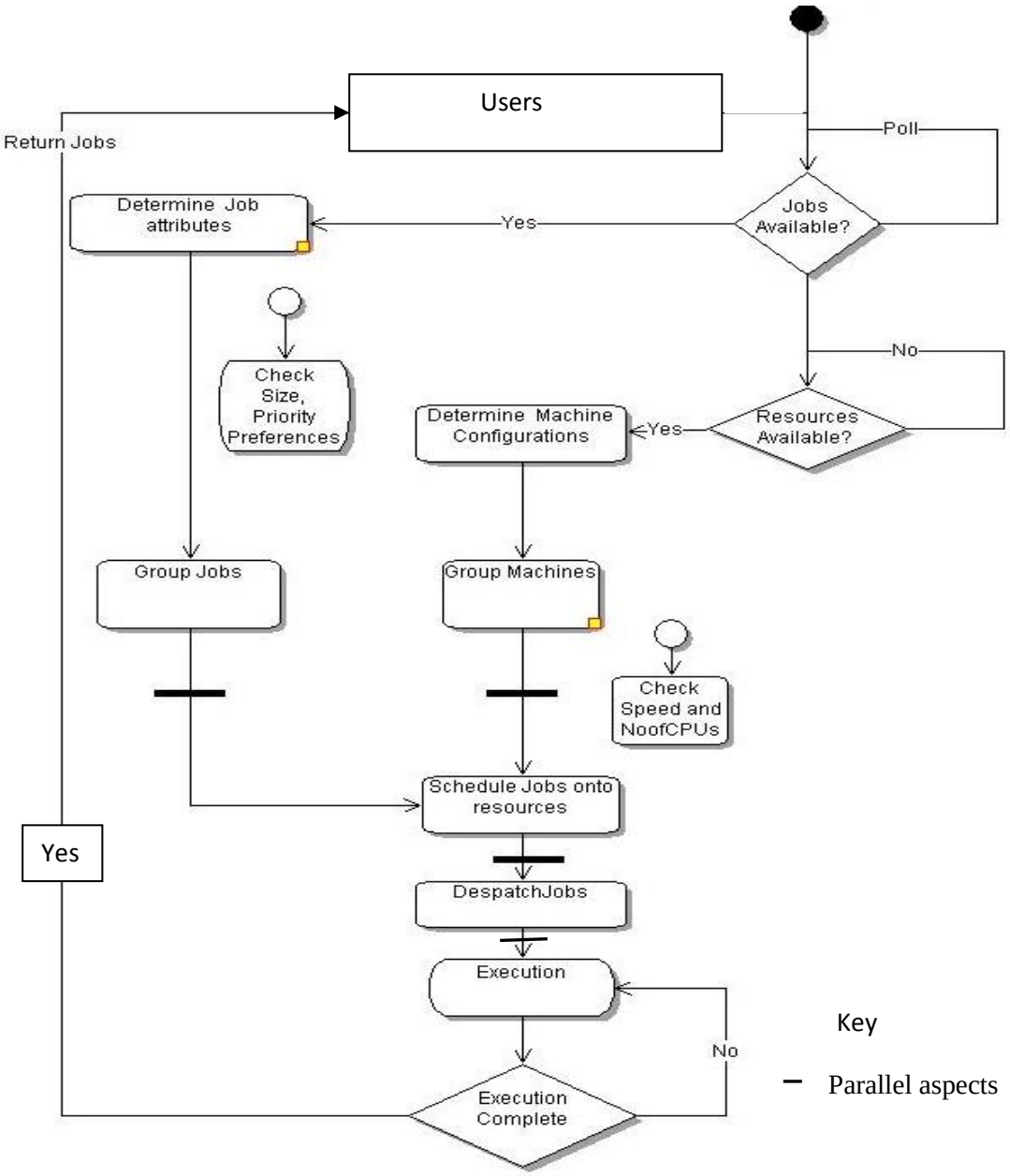


Figure 6: Activity diagram for the GPMS system

4.2.6 Sequence Diagram

The sequence diagram shows the timing and ordering of message interaction between the system and actors, external devices and external systems. The sequence diagram for the GPMS shows how a user submits his jobs and retrieves his jobs from the system. It also shows how the system handles the operations of sorting jobs, scheduling jobs and returning results to users. Figure 7 provides a sequence diagram showing the timing and interaction between users' and the system.

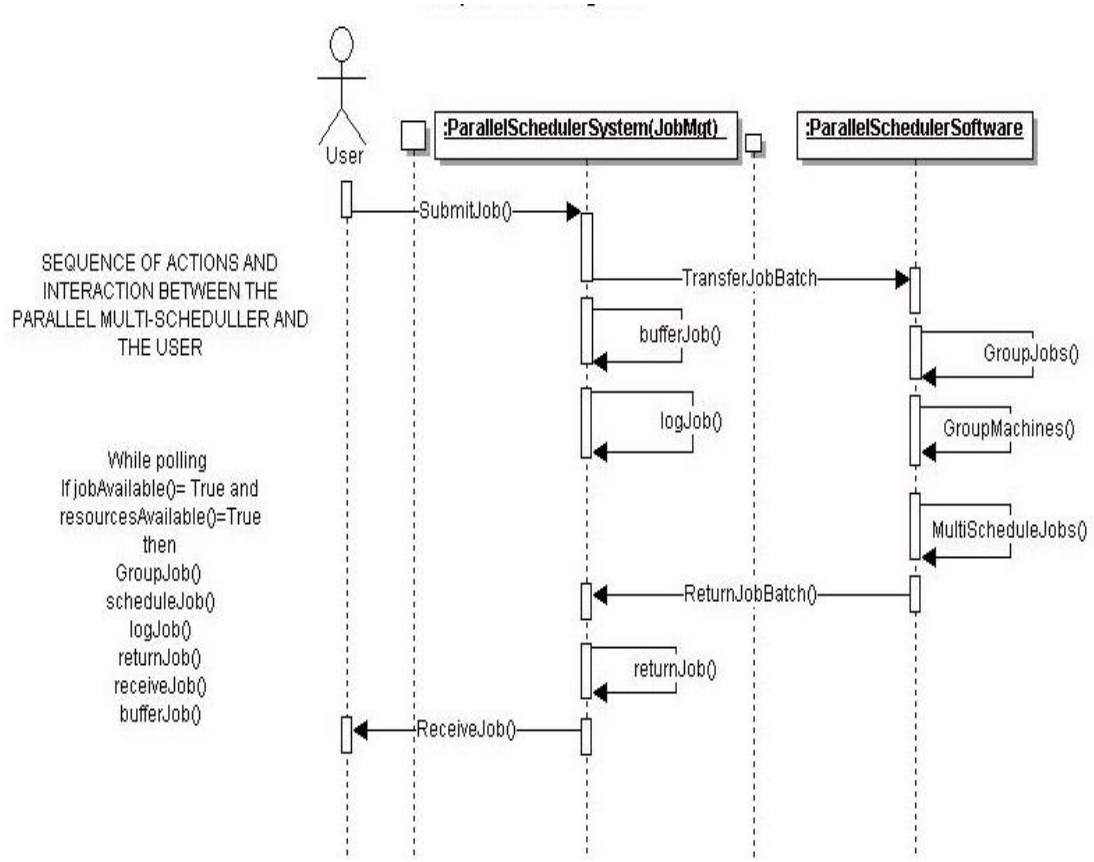


Figure 7: Sequence diagram for the GPMS system

4.2.7 Class Diagram

The class diagram is used to visualize the components of the system and define the classes, functions and attributes of the system. It also aids the coding of the system. Figures 8a and 8b show the class diagram for the GPMS system.

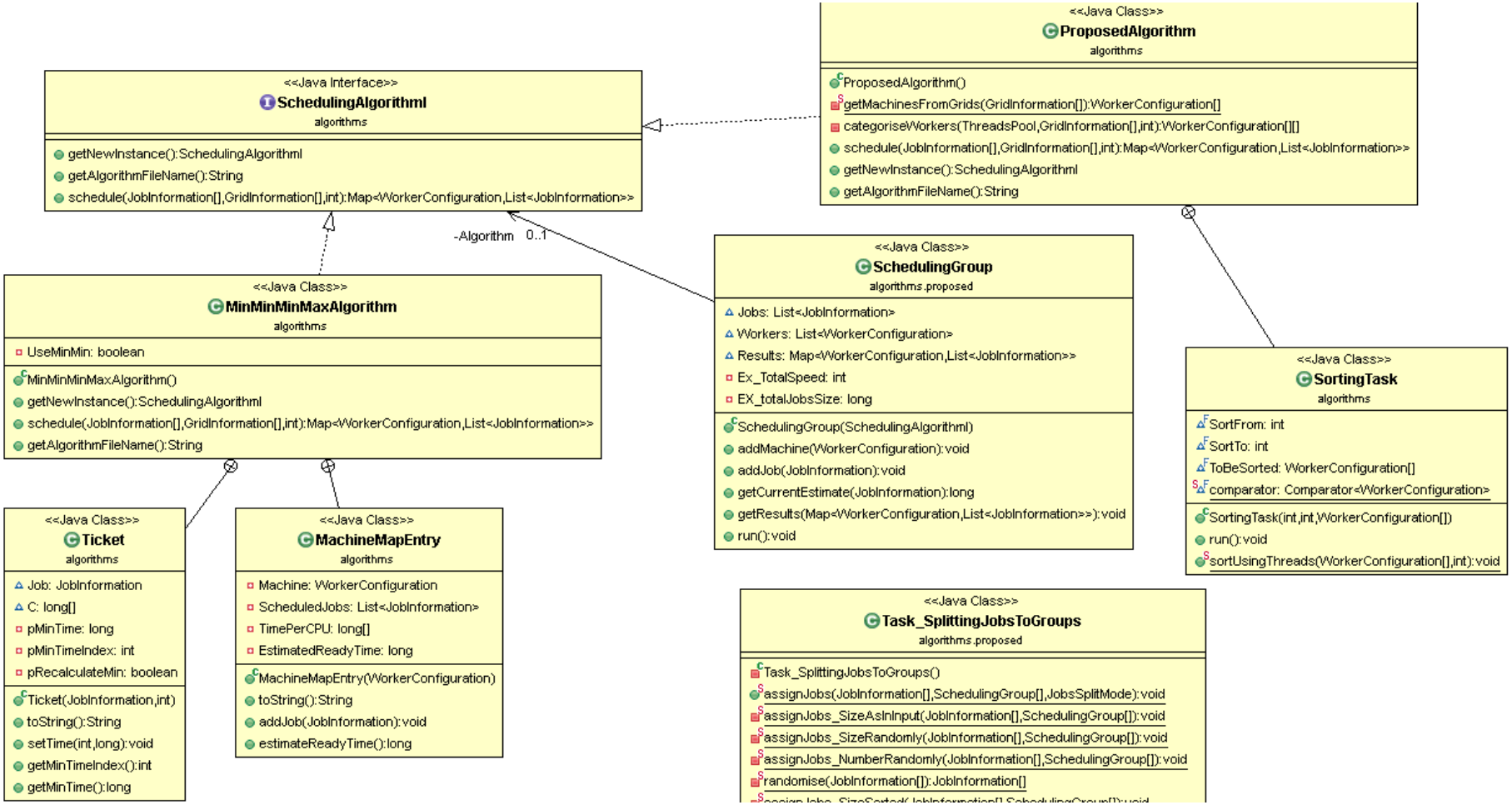


Figure 8a: Class diagram for the GPMS system

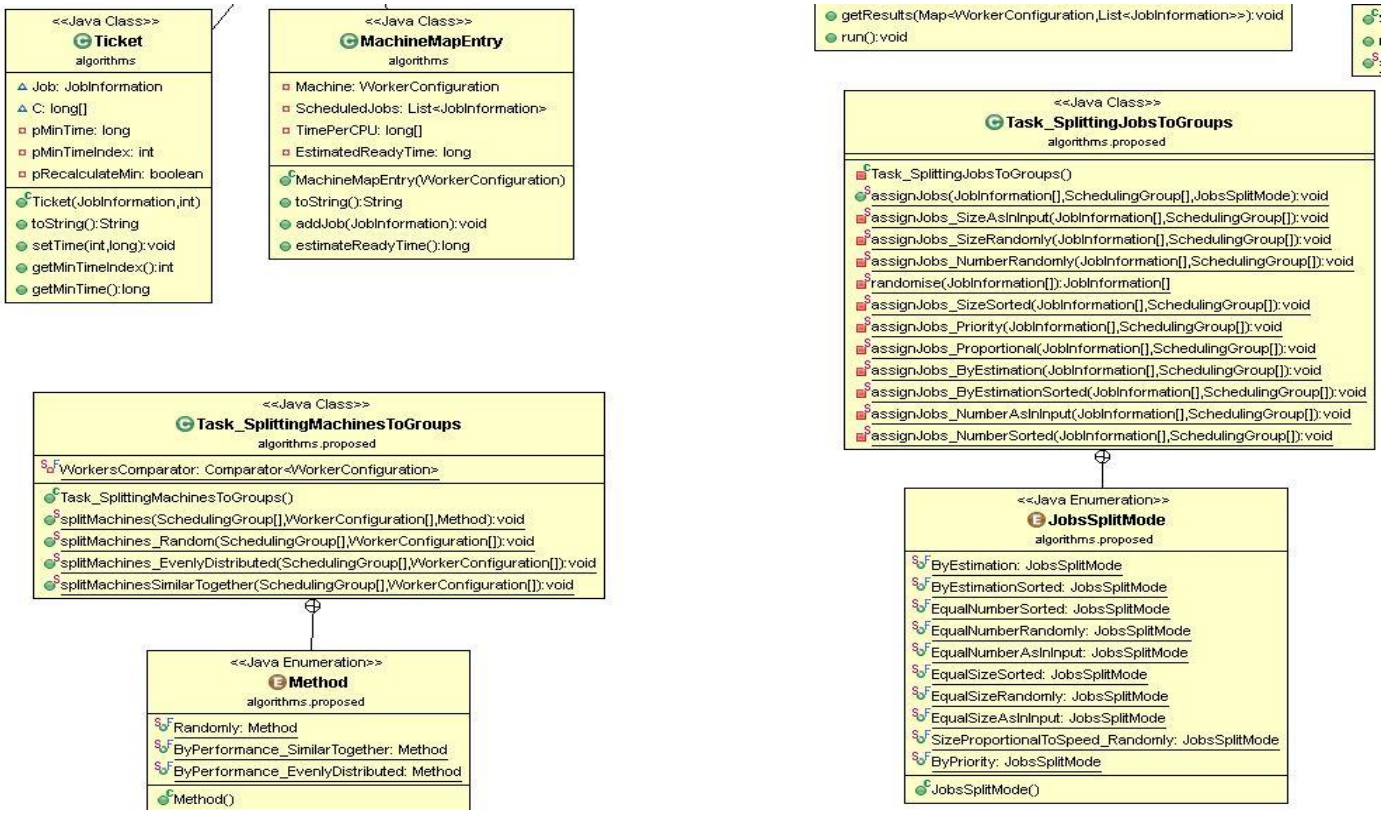


Figure 9b: Class diagram for the GPMS system

4.3 The GPMS

4.3.1 Overview of Processing

The GPMS is the general system that incorporates all the functions required of the system. Users' jobs are jobs generated and submitted by users for execution on the Grid. They are composed of distinct characteristics/attributes which are used to determine how jobs are sorted or categorized or where jobs are scheduled to. Jobs are scheduled onto Grid machines at Grid sites by the GPMS. The GPMS executes on multicore systems, hence explores parallel programming methods that utilize multicores to advantage. The system implements a dynamic thread pool as a multi-threading mechanism that controls the number of threads used for each execution. Controlling the number of threads for executions on multicores enhances the utility of the underlying multicore and provides parallelism. The threads are stored in a pool and used when required and released when not required. The threads are targeted to execute the scheduling algorithm within the paired machine-job groups. In the experiment, the number of threads was varied from one to sixteen for each group but in the analysis, we presented the points where the number of groups and threads are equal. Hence, two groups used two threads, four groups used four threads and eight groups used eight threads. Each thread executes the same scheduling algorithm within a group. Figure 9 shows a model depicting the GPMS.

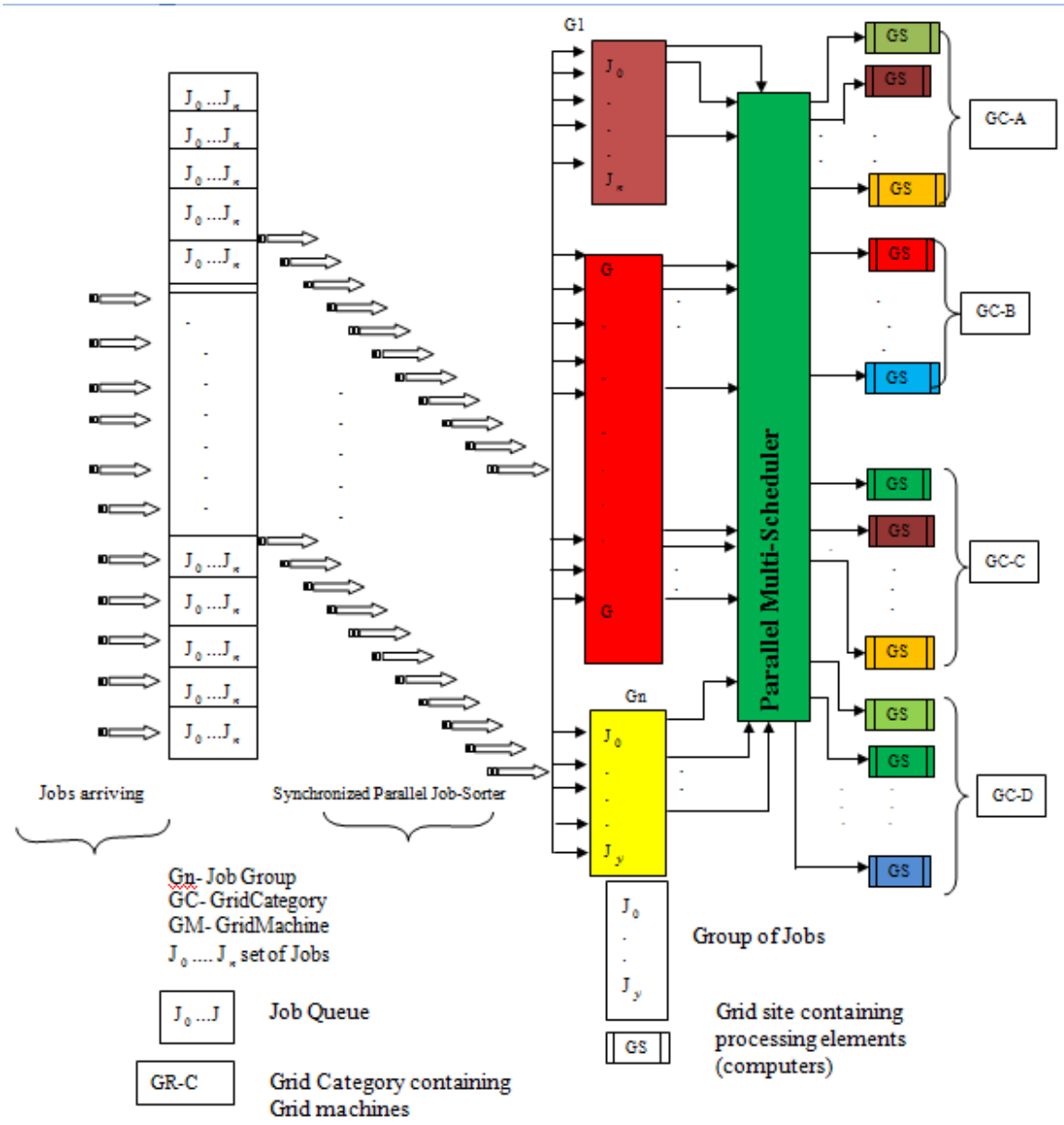


Figure 10: A model of the GPMS with multiple groups

Another way of looking at the GPMS is shown in Figure 10, this time with four groups. Jobs arrive from several sources. Jobs are sorted in parallel into groups based on the attributes of the jobs. They are then multi-scheduled in parallel from the various groups onto Grid machines at Grid sites. The Grid machines are also categorized into groups

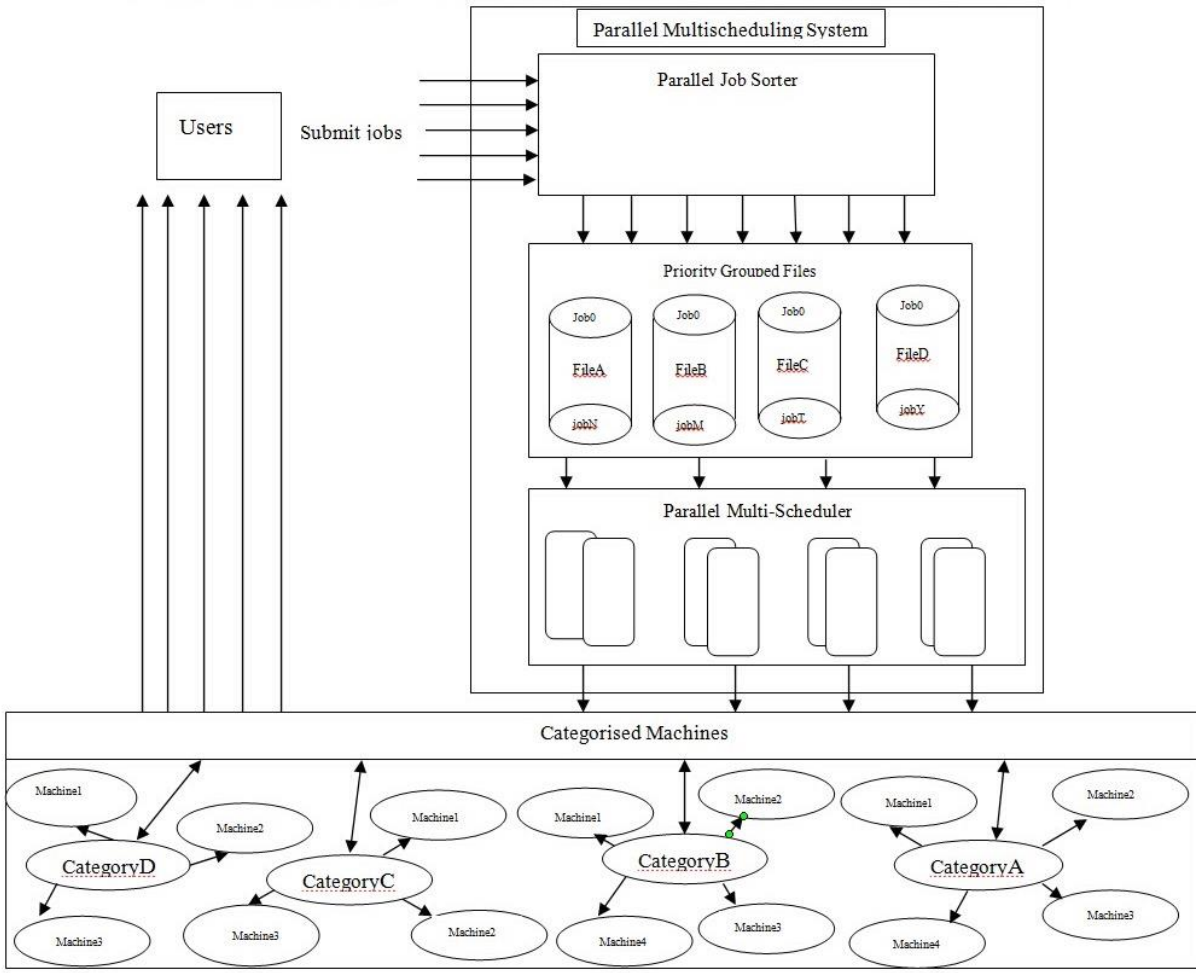


Figure 11: A model of the GPMS with four groups

4.3.2 GPMS Job and Machine Grouping

The GPMS requires jobs and machines to be grouped. The machine and job groups are then paired and scheduling occurs in parallel within the groups – the pairing between job groups and machine groups ensures independent and parallel scheduling within the groups. The scheduling algorithm used inside the groups is MinMin. Three different methods were implemented for job grouping in the GPMS model. These were the Priority, the ETB and the ETSB methods. Two methods were used for the grouping of Grid resources (machines). These were SimTog and EvenDist. After the grouping of machines and jobs separately, a pairing is made between job groups and machine groups. Then using multiple threads (multithreading), a scheduling algorithm (MinMin scheduling algorithm) is executed independently within the paired groups in parallel. A thread pool is created to enable parallel

scheduling within the groups. The method can achieve improvements in scheduling efficiency by approximately g times where g is the number of groups used, although overheads make an exact g times improvement unachievable.

Multi-threading was implemented with a dynamic thread pool. Threads were activated when needed and deactivated when no longer needed. The threads were varied from 1 to 16 in steps of power 2 (2^n) and groups varied between 2, 4, 8 and 16. This setting is deliberate because multicore computers exist in that order. Furthermore, it is important to observe the relationship between the numbers of groups used, number of threads used and number of cores used. In the analysis, we presented the points where the number of groups and threads are equal. Hence, two groups used two threads, four groups used four threads and eight groups used eight threads. Each thread executes the same scheduling algorithm within a group.

The number of groups and threads are specified by a GPMS administrator. How jobs are read into the system and how the jobs are grouped before scheduling is presented in the algorithm in Table 6. At present an automated system for determining numbers of groups and threads has not been developed but this could be part of future work.

Table 6 Algorithm for the GPMS

Step1. Start *Step2: Specify number of threads to use (this is set by the user)* *Step2: Specify number of groups to use* *Step3: Read jobs into the scheduler* *Step4: Read machines* *Step5: From the job attributes; estimate the priority, size, execution or completion time for each job* *Step6: Group jobs into number of specified groups(three methods are used)* *Step7: Group machines into the specified number of groups(two methods are used)* *Step8: Execute the scheduling algorithms within the groups using the inside groups scheduling algorithm – i.e. MinMin* *Step9: Write results to output file* *Step10: Stop* Results include total time of scheduling of jobs for job grouping method used, number of threads used, number of groups used, and number of jobs.

The GPMS splits jobs and machines into groups before executing the scheduling algorithm (MinMin) within the groups. Jobs are split (grouped) based on the estimated execution time computed from their size or priority if the Priority method is used. Jobs are initially held in a table which also holds their estimated size or priority. Three methods are employed in splitting jobs into groups:

Priority*:* Jobs are grouped based on priority. The resulting groups may not be balanced.

***Execution Time Balanced (ETB*)**: Jobs are grouped according to their execution time and balanced into groups.

Execution Time Sorted and Balanced (ETSB)*:* Jobs are balanced into groups according to their execution time but are first sorted from largest to smallest. This grouping algorithm then ensures jobs are more evenly balanced across groups in terms of their size.The resulting job groups contain sets of Grid jobs submitted by users but sorted into groups based on some characteristics from where they maybe scheduled to machine groups independently. The machine grouping methods used are:

Similar Together (SimTog) – machines of similar configurations are grouped together

Evenly Distributed (EvenDist) – machines are evenly distributed with regard to their configuration

In the next sections more detail is given on the job grouping and machine grouping methods.

4.4 Job Grouping Methods

Three job grouping methods were used: the Priority, the Estimated Time Balanced (ETB) and the Estimated Time Sorted and Balanced (ETSB) methods.

4.4.1 Design of the Priority Method

The Priority method differs from other methods in that it uses just four groups each for machines and jobs. The method categorizes jobs into four priority groups. Grid machines are equally distributed into four groups based on their configurations using two methods – SimTog and EvenDist. Machines and job groups are then paired before job scheduling is executed simultaneously and in parallel among the job-machine group pairs using the

MinMin algorithm. Scheduling of prioritized jobs from groups is targeted directly at the processors within the machines in the groups.

In the experiment, the Priority method uses the number of processors requested by the user (ReqNProcs) to determine the priority. Before submission, a user either states or selects the number of processors he needs his job to be executed on. The choices vary from not specifying (zero) to more than a few hundred. As provided in the Grid Work Flow archive, this attribute (ReqNProcs) may reflect the importance with which a user ascribes his job. Grid users who specify a higher number of processors for the execution of jobs could be regarded as desiring a higher priority for their jobs.

Grid jobs are characterized by many attributes and the choice of attribute(s) used to determine priority in this research is somewhat arbitrary; other attributes could have been used. For instance, in some applications, the choices made directly by the customer could be used to determine priority (Albodour, James and Yaacob 2012). In production environments, suitable attributes would be determined depending on available meta-data. In commerce, attributes can be determined by customers need, demand or cost (Buyya, Abramson and Giddy 2000), or by availability (Abraham, Buyya and Nath 2000).

The Priority method employs four priority groups from Priority Group 1 to Priority Group 4. Jobs in Priority Group 1 have the highest priority and those in Priority Group 4 have the lowest priority. In a system which groups machines such that machines are grouped according to performance, the Priority method can be used to ensure high priority jobs are mapped to high performing machines. In the GPMS, two methods of job grouping are used, SimTog and EvenDist. The idea behind the Priority method was that the priority of job groups would be matched to the priority of machine groups. This requires a machine grouping method like SimTog which puts machines of similar characteristics together. Hence jobs of Priority 1 would be matched with the machine group which contains the most powerful machines. However to work well, the incoming jobs must be evenly distributed in terms of priority and this cannot be guaranteed. If the EvenDist machine grouping method is used, all machine groups would be similar in power and in this case a priority match between jobs and machines would not be possible. In the analysis described in Chapter Five, it is shown how the machine grouping can affect the performance when the Priority method is used.

Muthuvelu et al. (2005) used (MI) million instructions or processing requirements of a user job to relate to the size or processing requirement of the job. The GPMS uses job size (which was computed by multiplying ReqTime by ReqNProcs). ReqNProcs (the number of processors requested by the user) was used because a job requiring one processor and another job requiring ten processors to execute certainly have different priorities (importance) set by their owners. Also, ReqTime was used because the resource time a user wants his job done also signifies the priority (urgency) with which the user attaches to his job. Hence, multiplying the requested time and the requested number of processors is a good way to quantify or represent the job size.

The Priority grouping method uses four groups. The Priority of jobs is determined by the number of processors requested by the user. Hence four categories: Very High, High, Medium and Low were chosen to conform to the number of groups used by the method.

The rule to assign the priorities is as follows:

If (ReqNProc is less than or equal to1) then JobPriority = Low;

If (ReqNProc is less than or equal to 2) then JobPriority = Medium;

If (ReqNProc is less than or equal to 4) JobPriority=High;

If (ReqNProc is greater than 4) then JobPriority=VeryHigh;

Table 7 shows the algorithm to determine the priority of jobs and allocate them to priority groups based on the number of processors requested by the user and Table 8 shows the steps taken to schedule a job using the priority method.

Table 7 Algorithm for the Priority method

Step1: Start
Step2:Establish 4 job groups (one per priority)
Step3: Accept next job
Step4: Assign priority to the job based on the number of requested cores (1- low, 2-medium, 3-4 –high, >4 –very high)
Step5: Add the job to the group with matching priority
Step6: Repeat Step3 to Step 4 until all jobs assigned to groups
Step7: Stop

Steps taken to schedule jobs using the priority grouping method and measure time are as follows:

Table 8 Scheduling steps using the Priority method

Step1: Start
Step2: Specify number of groups to use
Step3: Split jobs into groups based on their characteristics (Priority) - four priority groups were used in the Priority method but a different number of groups could be used.
Step4: Machines are split into groups based on their configurations- the same number of group is used as that for splitting jobs into groups.
Step6: Start scheduling clock to record time
Step7: *Execute the* (*InsideGroupsScheduling algorithm)* - (the *MinMin algorithm is used)*
Step8: Write results to files
Step9: Stop clock
Step10: Stop

Figure 11 depicts a simplified version of the flowchart for sorting of jobs to priority groups. It considers the priority, computational requirement and time requirement of the jobs as attributes. In this simplified version, the sorting is done using the number of processors requested to determine the priority of job.

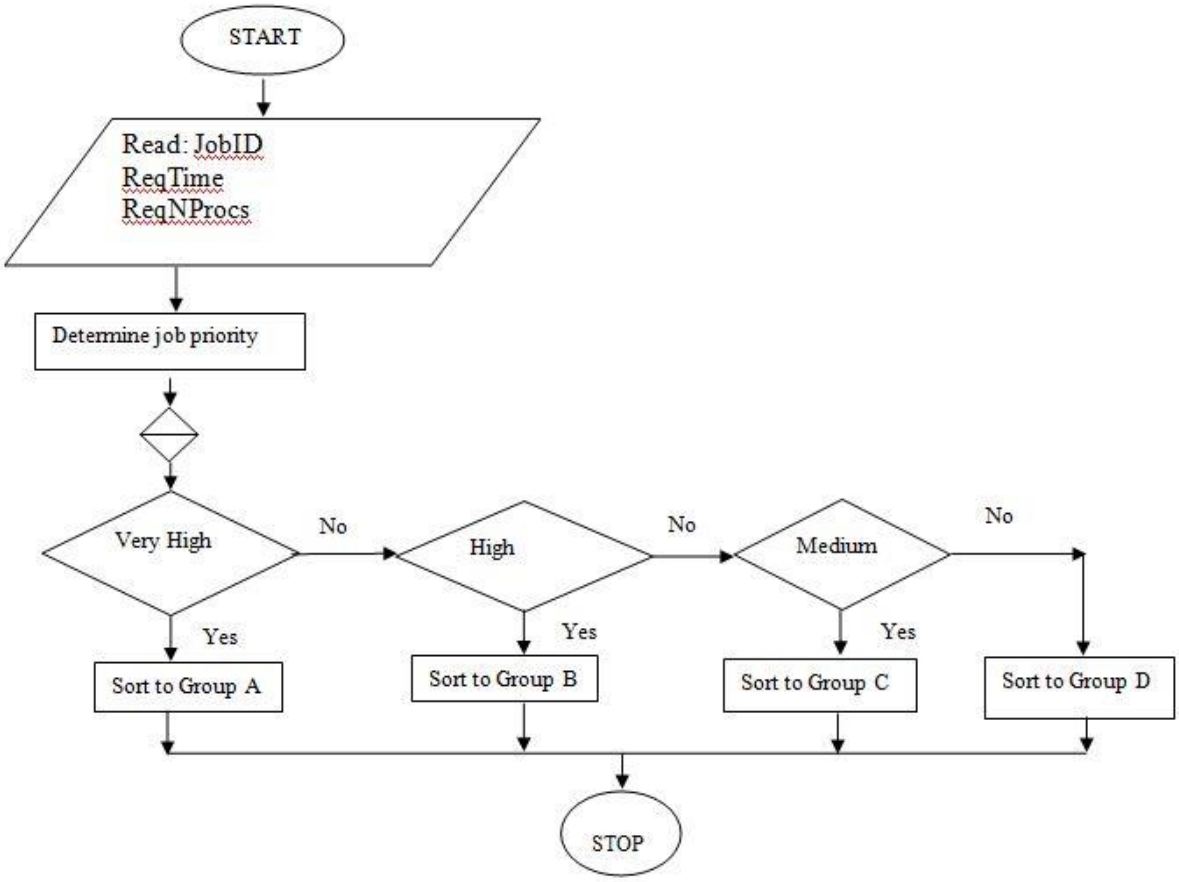


Figure 12: Flowchart for priority sorting of jobs

4.4.2 Design of the Execution Time Balanced (ETB) method

This section discusses the design of the ETB method of the GPMS model. One major difference between ETB method and the Priority method is that the ETB uses a method that varies the number of groups of machines and jobs.

The method uses an estimation of the processing time for each job to group the jobs. It attempts to even out the total processing times in groups by adding the next job to the group with the current lowest total processing time. The method takes jobs one by one and inserts the job into the group that can execute it fastest (including the time needed to process jobs already added to that group). First, jobs are read in and execution time of each job is estimated with reference to a base machine. Jobs with estimated execution times are then recorded in an Estimation table. The method accesses the Estimation table and groups jobs based on the estimated time of each jobs in the group. When a job is selected for grouping, the estimated execution time for the jobs is known and the total estimated execution time for

the group is also known. The job is grouped (sorted) to the groups with the *best* or *lowest* ***totalestimatedTime).*** The estimated execution times for the job is then added to the group with the lowest execution time, then the total estimated time for that group is updated and the next job is selected. This method ensures that the jobs are distributed fairly to all groups. The selection is repeated until all the jobs are allocated to scheduling groups before the real scheduling is executed from the groups. Table 9 shows the ETB algorithm.

Table 9 Algorithm for the ETB method of grouping jobs

Step1:	Start
Step2:	Select job from the Estimation table
Step2:	Select the group with the smallest **totalestimatedTime**
Step3:	Add job to group with the smallest **totalestimatedTime**
Step4:	Update the **totalestimatedTime** for the group
Step5:	Repeat until end of table
Step6:	Stop

4.4.3 Design of the Execution Time Sorted and Balanced (ETSB) method

This section discusses the design of the ETSB method of the GPMS. This method also differs from the Priority method because it can vary the number of groups of machines and jobs (like the ETB method).

The method first sorts jobs based on the estimated execution times before applying the ETB method to distribute jobs into the groups. Jobs are first read in and the estimated execution time for each job with reference to a base machine is generated and recorded in the Estimation table. Jobs in the table are then sorted based on their estimated execution time. Sorting is done in descending order and the job with the largest estimation time placed at the top of the list and that with the least completion time placed at the bottom of the list. Then starting from the biggest or top, the method takes jobs one by one and inserts into the group

that can execute it fastest or the group with the smallest **totalestimatedTime** and the **totalestimatedTime** for that group is updated accordingly. Just as in the ETB method, the jobs are added to the group with the lowest or best **totalestimatedTime** and the group is updated before the next job is picked and the process is repeated until the end of the Estimation table. The sorting employed in this method ensures that larger jobs are allocated before smaller jobs. Also, this method helps to increase the chance of all groups getting a fair share of the workload.

This method is similar to the ETB method except that jobs are first reordered or sorted based on the execution times before inserting them into groups. The largest jobs are placed at the top of the list; the method ensures a fairer balance across groups. Table 10 shows the algorithm for the ETSB method.

Table 10 Algorithm for the ETSB method of grouping jobs

Step1:	Start
Step2:	Sort jobs in the Estimation table
Step3:	Read next job from the Estimation table
Step2:	Select the group with the smallest **totalestimatedTime**
Step3:	Add job to group with the smallest **totalestimatedTime**
Step4:	Update the **totalestimatedTime** for the group
Step5:	Repeat until end of table
Step6:	Stop

Both the ETB and ETSB methods ensure that jobs are distributed equally among the groups. They also allow the number of groups and threads employed to be varied in each execution.

4.4.4 Job Attributes and Job Categorization

This section offers some observations on job attributes and job categorisation and how these are used in this research.

The attributes of a job are distinct characteristics that distinguish jobs and determine how jobs are grouped then scheduled. The attributes of a job also determine a job's priority. Other than size and computational requirements of the jobs, there are other options available for Grid users to **specify options** which could still be useful in categorizing jobs. These include:

Trust - the issue of trust is based on a prior knowledge or long term use of a particular Grid resource or recommendation from friends, their preference for such trusted Grid site to execute their job.

Budget – this is related to how much users are willing to pay to get their jobs executed, such options become useful when budget becomes the match-making criteria. This option determines between the highest and lowest budget. The budget is a very critical attribute for some (economic model) schedulers as they determine a match between users' jobs and the Grid site.

Time Requirement or Deadline– this factor is important when time is the most critical issue factored into the scheduling need of users.

Users of the Grid are provided with means to specify options when submitting their jobs. These options and attributes are used by the scheduler for scheduling decisions. This work excludes those options but concentrates on the attributes present (in the source file) and relevant for this research.

4.5 Machine Grouping

The two methods used to split machines into groups are Similar Together (SimTog) and Evenly Distributed (EvenDist).

4.5.1 Design of SimilarTogether (SimTog) Method

This method uses the configuration or performance attributes of machines like the number of CPUs and speed of the CPU to group them together. Machines are first sorted by performance (Number of CPUs*SpeedofCPU) from slowest to fastest. The entire list of machines is then split into g groups represent the number of groups to be used for the execution. The first N machines are added to the first group, the next N machines are added to the next group and the process is continued until all machines are added and g groups are formed. As a result, the first group is guaranteed to have the slowest machines, followed by the next in that order. The last group is guaranteed to get the best set of machines.

This means some groups have better performing machines than others. Groups with better machines (Number of CPUs*SpeedofCPU) may complete their execution faster than groups with slower machines (Number of CPUs*SpeedofCPU). And the group with the least (Number of CPUs*SpeedofCPU) ranking will perform poorly compared to the other groups if same tasks are assigned to all groups. It will be advisable to assign higher priority group jobs to higher configuration machine groups and lower priority job groups to lower configuration machine groups. This will ensure some load-balancing, improve overall execution time and ensure some level of QoS. However Priority is only one of the methods used within the GPMS. Table 11 shows the algorithm for splitting of jobs using the Similar Together (SimTog) method.

Table 11 Algorithm for the SimTog method of grouping machines

Step1: Start
Step2: Sort machines based on configurations (i.e. number and speed of processors)
Step3: Determine g (g is the number of job groups)
Step4: (Integer)-Divide number of jobs by g giving N
Step5: Add top N machines to the first group
Step6: Add next N machines to the next group
Step7: Repeat Step7 until all machines are assigned
Step8: Stop

4.5.2 Design of EvenlyDistributed (EvenDist) Method

This method eliminates the immediate inadequacies in the SimTog method by ensuring that the various machines are equally or at best equally split and distributed into the groups (based on their configuration or performance specification). This method guarantees that each group has similar machine configurations. First, machines are sorted based on configuration or performance (Number of CPUs*SpeedofCPU) from slowest to fastest. Then the first machine is added to the first group, the second machine to the second group, then third to the third group and fourth machine to the fourth group. The process is then repeated until all machines have been allocated. This method provides a more balanced processing infrastructure which might suit some input job sets better than SimTog. Table 12 shows the algorithm for EvenDist method.

Table 12 Algorithm for the EvenDist method of grouping machines

Evenly Distributed Method
Step1: Start
Step2: Sort machines based on configurations (i.e. number and speed of processors)
Step3: Register number of groups
Step4: Add first machine to first group
Step5: Add next machine to next group
Step6: Repeat Step5 until last group is reached.
Step7: Add next machine to first group
Step8: Repeat Step5 and Step6 until all machines are assigned to groups.
Step9: Stop

4.6 Experimental Testbed and Simulations

This section presents the experimental test bed and simulation methods used to evaluate the GPMS. First the simulation of the Grid is discussed and then a description is given of the job input source file and the attributes that were relevant to the evaluation, particularly with regard to simulation of execution time.

Simulations were used rather than actual machines test on Grid sites because of the difficulty in accessing the real Grid system. Simulations were made of Grid sites, Grid machines, CPUs and execution times of jobs on the machines.

4.6.1 Grid Site

The Grid is composed of an aggregation of Grid sites that are distinguishable from others due to their peculiar differences like owners and policies. Grid sites are composed of several Grid resources or processing elements with varying configurations controlled by owner policies. The computing machines within each Grid site are unique with their distinct characteristics or attributes. A Grid site can contain any number of compute resources.

Muthuvelu et al. (2005) in their study characterised Grid resource with: resource ID, name, total number of machines in each resource, total processing elements (PE) in each machine, MIPS of each PE, and bandwidth speed. The GPMS characterised the Grid with Grid ID (GId), the network bandwidth or speed of the network connection (n.b - Four categories were used to categorise network bandwidth in order to match the four priority groups used in the priority based grouping method**)**, and the number of computing machines. The features of machines in the Grid were characterised differently because the machines in a Grid site are distinct and different from each other. The GPMS characterised Grid machines with machine ID (Mid), number of processors, speed of processors and RAM size. Table 13 shows the features and characteristics of Grid site used in the simulation experiment.

Table 13 Features and characteristics of a Grid site

Features	Characteristics	Attributes
Network Bandwidth	Every Grid site is connected to the Grid via a network and the speed of the network connecting the Grid site determines to an extent the performance of the Grid. The network bandwidth (NBW) or speed of a Grid site is therefore used as one of the attributes to characterize a Grid site.	Network bandwidth or speed (NBW) are sub categorized into; Super-Fast (SF), Very Fast (VF), Medium Fast (MF) and Not Fast (NF) with weights 4, 3, 2 and 1 respectively.
Number of Machines	This feature simply refers to the number of computers that the Grid site contains. The number of machines within a Grid site can be arbitrary. It can be any number and in some cases due to computer system characteristic of failure and repair, the number can vary from time to time.	This number varies over time hence there is no need for categorization
Grid ID	This is the identification features of the Grid site. The Grid ID can be the name or number used to identify the Grid	Name or number or combination of both

4.6.2 Grid Machines

Grid machines are the computing resources that make up a Grid site. Every Grid site contains hundreds to thousands of computing machines, and each computing machine is distinct by its configuration. Grid machines or compute resources are characterized by distinct features like the machine's identification (MId), speed of processor (SP), number of processor cores (NPC) and RAM size. Table 14 shows the features of Grid machines used in the simulation experiment.

Table 14 Features and attributes of a Grid machine

Features	Characteristics	Measure
Machine Identification (MId)	Used to identify an individual machine	n/a
The Number of Processor cores (NPC).	The number of processors within a machine can determine how efficiently that machine can execute jobs. The number of processors contained within a machine is therefore a characteristic feature of the machine	The number of processors contained in a Grid machine can vary from 1 to n, where n is the number specified
The Speed of Processor (SP).	The speed of processor of a machine determines how fast a job can be executed on a machine. This attribute is also used to determine to which group a machine is categorized.	The speed of processors are rated in MHz or GHz
The Ram Size	RAM is the part of memory where jobs in execution are held within each machine and plays a major role in determining how many jobs are executed over time. Large RAM sizes determine the size of jobs that can be resident in memory while in execution. It also determines how many jobs can be executed within the memory at the same time.	This attribute RAM size is measured in MB or GB

4.6.3 Simulation of Grid, CPU Speed and Number of Cores

The Grid was simulated to be characterized by the following attributes: Category; CPU; RAM; Bandwidth. For example {A; 1200; 2000000; 1000} represents Grid site A, CPU 1200, RAM 2000000, and Bandwidth 1000.

The computer machine was defined with the following attributes: CORES; CPU; RAM. For instance {2; 2000; 2000000} represents a Grid resource (machine) with 2CPUs, 2000 MHz (2GHz) and 2000000B (2MB). Table 15 shows the characteristics of the simulated Grid and Figure 12 is a schematic diagram of the GPMS and illustrates how users access the Grid. Users' jobs are submitted to the Grid. The jobs are then sorted in parallel into groups from

where the multi-scheduler schedules them to Grid sites for execution. Results are returned to users after execution is complete.

The Grid attributes discussed within this section are utilized by the GPMS in its scheduling decisions

Table 15 Characteristics and components of the simulated Grid

Grid Site	**Characteristics**			**Grid Site**	**Characteristics**		
	Number of machines	Speed of CPU	Number of CPU/ Cores		Number of machines	Speed of CPU	Number of CPU/ Cores
A	30	1GHz	1	C	60	1.5GHz	2
240	30	2GHz	1	480	60	2GHz	2
Machines	30	3GHz	1	Machines	60	3.5GHz	2
	30	4GHz	1		60	4GHz	2
	30	1GHz	2		60	1.5GHz	4
	30	2GHz	2		60	2GHz	4
	30	3GHz	2		60	3.5GHz	4
	30	4GHz	2		60	4MHz	4
B	50	1.5GHz	2	D	50	1.5GHz	2
400	50	2GHz	2	600	50	2GHz	2
Machines	50	3.5GHz	2	Machines	50	3.5GHz	2
	50	4GHz	2		50	4GHz	2
	50	1.5GHz	4		50	1.5GHz	4
	50	2GHz	4		50	2GHz	4
	50	3.5GHz	4		50	3.5GHz	4
	50	4GHz	4		50	4GHz	4
					50	1.5GHz	8
					50	2GHz	8
					50	3.5GHz	8
					50	4GHz	8

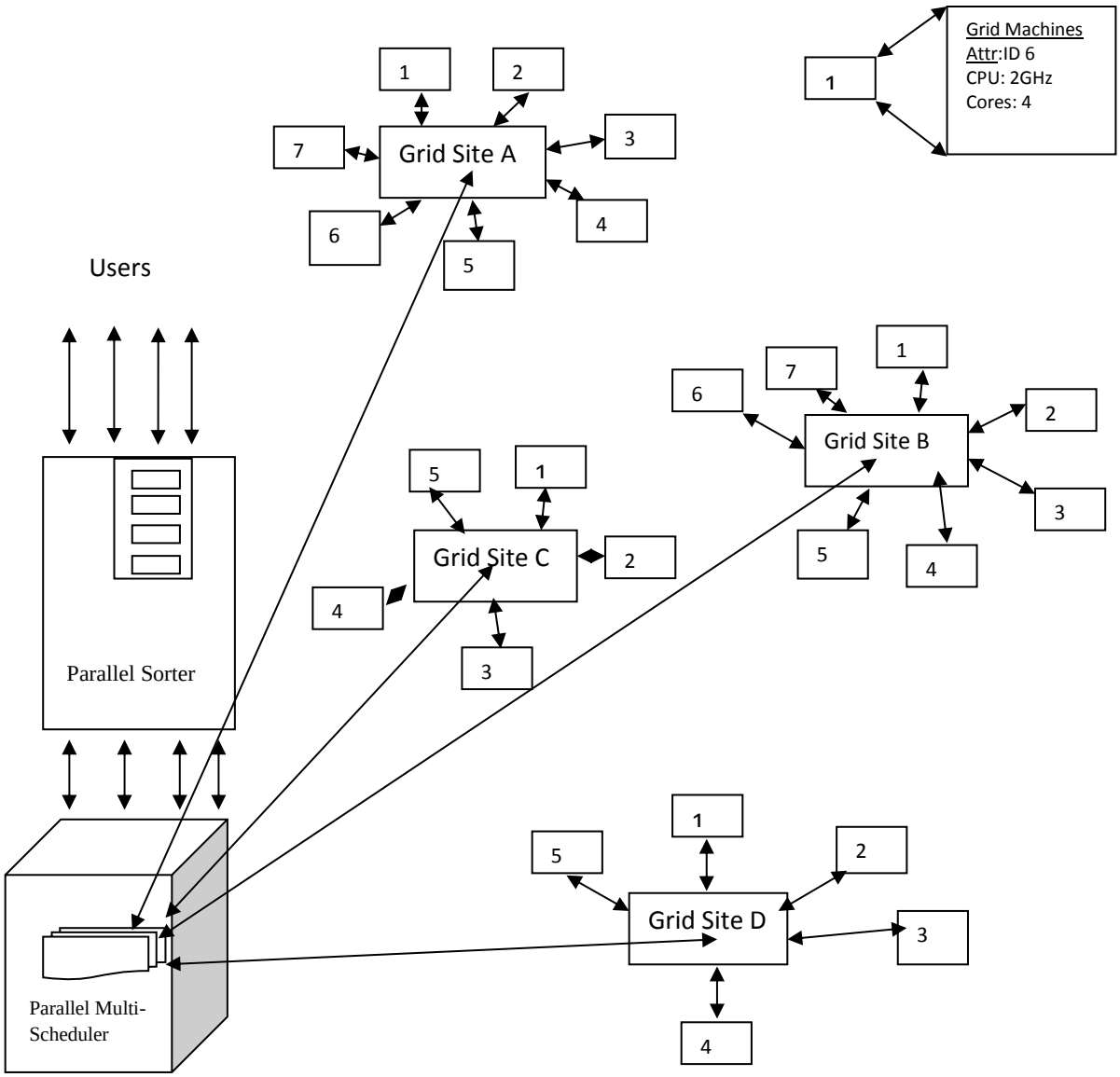


Figure 13: Schematic diagram of the system

4.6.4 Local Policy

Within every Grid is a local user policy that determines how resources within the Grid site are utilized by either incoming jobs or jobs from within the Grid site. Some policies are tailored to service jobs coming from outside the Grid site; other local policies are designed to favour jobs from within the site; while others try to strike a balance between the outside jobs and inside jobs.

Some policies are dynamic and can self-adjust to favour outside jobs when the internal nodes are not busy (close of work) and re-adjust to favour the internal jobs when they come alive (at the start of work). This work does not consider the effects of local policies on scheduling but assumes that all machines at Grid sites are available and directly addressable by the multi-scheduler.

4.6.5 Source of Jobs to the System

When workloads are not available or do not represent the real usage of the system, there may be a discrepancy between the success of the system in theory and the success of the system in practice (Cirne and Berman 2001). Realistic workloads are critical for the design and analysis of computer systems (Chapin et al. 1999, Feitelson and Rudolph 1998 and Mache, and. Windisch 1998). Good sources of realistic workloads are logs that record the characteristics of jobs submitted to a production system (Cirne and Berman 2001).

The Grid Workloads Archive (Iosup et al. 2008) is designed to make traces of Grid workloads available to researchers and developers alike. It is comprised of data from more than nine well known Grid environments, with a total of more than 2000 users who have submitted more than 7 million jobs. The Grid Workloads Archive project has lasted well over 13 years spanning over 130 sites with over 10 000 resources. It contains files both in plain text format and the Grid Workload Format (GWF). The GWF file contains 29 attributes relating to the running of a job in a Grid. However, a very high percentage of the values are missing from some of the core fields (such fields that contain missing values are denoted with -1). These missing fields necessitated that some assumptions needed to be made in order to have sufficient input for the GPMS.

A Grid scheduler should have the capability to accept users' jobs as inputs which are Grid-enabled before submission. Once jobs are submitted, they are either stored in buffers as files (batch mode) or scheduled immediately (immediate mode). Grid systems are capable of utilising real-time, online and batched data. The GPMS uses batched jobs made available from the GWA site for experimentation but the system can be adapted for real-time or online data in real life situations.

The attributes of user jobs are the distinct items that characterize Grid jobs. Grid scheduling

algorithms depend largely on the attributes of jobs as specified for the optimization of the algorithm and delivery of quality of service. The experiments used job attributes which were relevant for the purpose. Attributes used for estimating completion time of jobs are shown in Table 16. The full header file and file format of the GWA is shown in Appendix A while Appendix B acknowledges the contributors to the various trace files.

Table 16 Selected attributes from the Grid Workloads Archive's trace file

Attribute	Description
ReqTime	Requested time measure in wall clock seconds
ReqNProcs	Requested number of processors
RunTime AverageCPUTime Used	Time job actually executes Average CPU time over all the allocated processors
NProcs	This is the number of allocated processors

ReqTime: This is the expected execution time estimated and provided by the user.

ReqNProcs (Computational / Processing Requirement):

This is the number of processors specified by a user for the processing or execution of a job at the point of submission. It determines the computation requirement or the processing needs of a Grid job. It is simply stated in numbers. This value is used in the simulation to determine which machines are able to process the job and may contribute to estimating job size.

RunTime: This is the actual execution time from when the job started to the time when it finished.

AverageCPUTimeUsed: This is the time actually used by the processor to execute the task averaged over the number of allocated processors.

NProcs: This is the number of processors allocated for execution

4.6.6 Simulation of Priority and Execution Time

Table 17 shows some typical values of the relevant attributes of some rows from the GWF trace file. The number of data items in the file is much larger than those shown in Table 17. However the attributes in Table 17 are the ones that were used to estimate priority and execution time in the simulation. A larger sample from the trace file is given in Appendix A.

Table 17 Example rows of values (relevant attributes only) from the GWF trace file

JobID	RunTime	NProcs	AverageCPUTime	ReqNProcs	ReqTime
0	0	4	-1	4	3600
1	19	1	-1	1	3600
2	10	5	-1	5	3600
3	8	90	-1	90	3600
4	19	100	-1	100	3600
5	25	1	-1	1	3600

4.6.6.1 Simulation of Priority

The Priority job grouping method needs a Priority metric for each job. The priority can be determined in various ways and can also be assigned directly by the user. In this research a Grid Work Flow archive (Iosup et al. 2007) was used as input to the experimentation. This archive did not include an explicit Priority measure. In the experimentation, the number of processors requested by the user (ReqNProcs) was used to determine the priority. This approach was somewhat arbitrary but it was sufficient for the simulation.

4.6.6.2 Simulation of Execution Time

Scheduling of jobs in Grid environment is challenging and requires optimisation of multiple variables. To achieve optimum schedule and proper resource utilization, the correct estimation of a job's execution time is vital. In some real systems, the user is required to

provide an estimate of the execution time of a job to enable better scheduing. The accurate estimation of execution time of jobs improves the efficiency of the scheduling algorithm, improves resource utilisation, helps to reserve resources in advance and also serves to meet some user QoS.

Estimating the execution time of jobs is a complicated task and has been the interest of many researchers (Liang et al. 2013, Quezada-Pina et al. 2012, Liu, Abraham and Hassanien 2010, Selvi et al. 2010, Franke, Lepping and Schwiegelshohn 2007, Tchernykh et al. 2006, Jeng and Lin 2005, Alem and Feitelson 2001, Braun et al. 2001, Ali et al. 2000, Hotovy 1996, and Tuomenoksa and Siegel 1981). In the GridSim experiment, Buyya and Murshed (2002) packaged jobs as Gridlets whose contents include the job length in MI (Million Instructions), the size of job input and output data in bytes along with various other execution related parameters. The job length is expressed in reference to the time it takes to run on a standard resource PE with (Standard Performance Evaluation Corporation) SPEC/MIPS rating of 100. The processing time of Gridlets is estimated based on 100 time units with a random variation of 0 to 10%. In another study, Tchernyk et al. (2006) and Franke, Lepping and Schwiegelshohn (2007) used the estimated execution time provided by the user at job submission as the execution time estimate of the jobs. In a related study, Liu, Abraham and Hassanien (2010) adopted a strategy to dynamically estimate the job lengths and estimate the completion time of jobs through load profiling, historical data or from some user defined attributes. This method is inadequate as most users' estimates have been found to be incorrect (Selvi et al. 2010) and imprecise (Quezada-Pina et al. 2012). Selvi et al. (2010) used rough set techniques to analyse the history of jobs and estimate the execution time of jobs. The method groups similar jobs and identifies the group to which the newly submitted job belongs and based on this similar group identified, the execution time is estimated. But estimates based on historic data cannot be very reliable as users jobs are dynamic and subject to change. Liang et al. (2013) implemented a method to evaluate execution time estimation for parallel jobs based on user behaviours in clustering of execution time estimation. By exploring the job similarities and revealing the user submission patterns, they used behavioural clustering of execution time to establish a pattern for users' jobs and used that to improve accuracy of overall job execution times. This method is also not very reliable as users' behaviour is dynamic and subject to change.

In the GPMS experimentation, execution times of jobs are computed based on actual traces from Grid workloads archive. The execution times of the jobs are simulated with the size of jobs computed from ReqTime and ReqNProcs. If these values are not provided, the GPMS system uses the AverageCPUTimeUsed (average of CPU time over all allocated processors) provided in the log entry to estimate the execution time of the jobs before scheduling.

The size of a job can be used to determine its processing or execution time – depending on its processing requirements. For instance if we multiple ReqTime by ReqNProcs we have some estimate of size. A more accurate estimate may come from AverageCPUTime multiplied by ReqNProcs but AverageCPUTime was not always available. A value not available is shown as -1 in the file. Because of missing values it was not possible to accurately replicate original job size from the trace file but some values available were used to generate a set of jobs with estimated sizes. Whilst recognizing that this approach was somewhat arbitrary, the estimations served the experiment adequately as a range of jobs of varying sizes with which to experiment was provided. Appendix A shows typical values of the attributes from some of the rows in the trace file and Table 18 shows the pseudo code for estimating job size.

Estimating execution time using file size and speed of the processors is easy and may seem one of the most feasible approaches (Xhafa and Abraham 2010) but the file size of jobs does not represent a true picture of the execution time of the job/file either. For instance some smaller jobs with several loops or iterations may take longer to complete than larger jobs without loops or iterations. In the same vein, some smaller jobs that require more I/O activities than larger jobs may take longer to complete due to slow processing activities caused by blocking during I/O request.

In summary, estimation of execution time is difficult and error-prone and short of extremely detailed analysis of code and data which is likely to negate any benefits of parallelisation, any system can only attempt best efforts based on job size (e.g lines of code), user specified requirements, previous history, user or job profiling and various level of code and data inspection.

Table 18 Pseudo code for estimating size of jobs

------- Pseudo code for job size----------	---------- Explanation ----------
Job.Size *- calculated as* *if(ReqTime != -1 AND ReqNProcs != -1)* *Size = ReqTime * ReqNProcs* *else if(ReqTime = -1* *Size = ReqNProcs* *else Size = AverageCPUTimeUsed*	If the ReqTime (requested time) is provided and ReqNProcs (requested number of processors) is provided, then size is the product of the two variables, else if ReqTime is not provided, then size equals ReqNProcs. Else if both variables are not given, then size is derived from the average CPU time used.

The simulation of the execution time was based on the job attributes provided in the file to estimate job size. Job size is used to estimate how long a job of size x will take to execute on a standard machine which is deemed as a 1 core machine with 1GHz processor. Table 19 shows the algorithm for the simulation of the execution time of the job on a particular machine.

Table 19 Algorithm for simulating execution time of jobs

SIMULATION OF EXECUTION TIME
Step 1: Start
Step2: Set the job size to be job execution time (T) on a reference machine (1 GHz, 1core)
Step3: Scale the expected time to match the current machine
Step4: Calculate performance ratio (R) between the current and the reference machine
Step5: Return the expected execution time divided by the performance ratio (T/R)
Step6:Stop

4.6.6 Executing Dynamically Generated Jobs

Scheduling of jobs without prior knowledge of the execution time of the jobs is referred to as non-clairvoyant scheduling. Quezada-Pina et al. (2012) noted that scheduling jobs with unspecified execution time is difficult, decreases the efficiency of the scheduling algorithm

and of the scheduler, as time is spent calculating (estimating) the execution time of jobs on machines.

The GPMS system deals with batched jobs and requires that a limited number of jobs are available before grouping of jobs can begin. If jobs are generated dynamically or received real-time from users, then based on the attributes provided by users, GPMS would gradually batch the jobs, computes the size or execution time of the jobs (based on the attributes provided), then, when the required batch number is reached, group the jobs before scheduling. Hence, if jobs are generated dynamically or if users' jobs are accepted in real-time, considerable time will be wasted while waiting for jobs to get to the limit for a batch. Also, job slowdown will be high if the number of jobs does not get to the limit on time.

To lessen the time wastage and reduce the slowdown, the job limit can be reduced to allow grouping and scheduling activities to take place more frequently. Future research could be to investigate how to efficiently combine batch scheduling with dynamic scheduling so that urgent jobs do not have to wait.

4.7 Experimental Design

This section discusses the experimental design and the platform of execution. The experiments employed the three job grouping methods in turn with each of the two machine grouping methods presented in section 4.5. Experimentation was carried out in phases. Seven different experiments were carried out, each of which consisted of a number of variations.

In the first instance, the MinMin scheduling algorithm was executed to schedule a range of jobs. This first experiment is treated as the base experiment and results from this experiment are compared against results from the other experiments.

The second and third experiments used the Priority job grouping method in combination with the two machine grouping methods (SimTog and EvenDist). The fourth and fifth experiments used the **ETB** method in combination with the two machine grouping methods. And lastly, the sixth and seventh experiments used the **ETSB** method in combination with the two machine grouping methods.

The experimentation was executed on one of Coventry University's HPC systems – known

locally as Pluto. The configuration of the HPC machine (Pluto) system on which the experiment was executed is as follows:

Number of physical CPUs per node/head: 2

Numbers of cores per one compute node/head: 12

CPU family: Intel(R) Xeon(R) CPU X5650 2.67 GHz stepping 02

Operating System: Linux x86_64 RHEL 5

In the experiment, a Grid environment was simulated comprising of four Grid sites each consisting of machines with different CPU speeds and number of processors. The parameters used in the experiment are the number of groups, number of threads used (varied from 1 to 16 in steps 2^n (n = 1 to 4) and the number of jobs scheduled ranged from 1000 to 10000in steps of 1000.

4.7.1 The Experiments

The various experiments are discussed in this section.

Experiment 1 – the Base Experiment

In the first experiment, the MinMin algorithm was executed on the HPC system to schedule a range of jobs (from 1000 jobs to 10000 jobs in steps of 1000). This was repeated using 1, 2, 4, 8 and 16 threads. In each instance of the experiment, the time of scheduling was recorded. Time of scheduling is the time taken to schedule each set of jobs, that is the time taken to schedule 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, and 10000 jobs in turn by each of the thread cardinalities. This experiment executes only the MinMin algorithm without employing the grouping method.

Experiment 2 – Priority Method 1 (uses four constant groups)

The second experiment used the Priority method to group jobs and the SimTog method to group machines before implementing the MinMin scheduling algorithm within the paired groups to schedule the same range of jobs as in the first experiment (1000 to 10000 in steps of 1000). For each instance of the scheduling execution, the time it took to schedule the range of jobs was recorded. This experiment used only four groups because there were four priority groups of jobs, while the number of threads was varied from 1 to 16 in steps 2^n (n = 1 to 4).

For each of the combinations, time taken to schedule and the makespan for each variation was recorded.

Experiment 3 - Priority Method 2 (uses four constant groups)

The third experiment used the Priority method to group jobs and the EvenDist method was used to group machines before implementing the MinMin algorithm within the paired groups to schedule same range of jobs (1000 to 10000 in steps of 1000). This experiment also used only four groups because there were four priority groups of jobs while the number of threads was varied from 1to 16 threads in steps 2^n (n = 1 to 4). For each of the combinations, time taken to schedule and the makespan for each variation was recorded.

Experiment 4 – ETB Method 1 (varied groups from 2, 4, 8 to 16)

The fourth experiment used the ***ETB*** method to group the jobs and the ***SimTog*** method to group machines before implementing the MinMin scheduling algorithm to schedule the same range of jobs as in experiment 1 above between paired groups of jobs and machines. Several runs of the experiment were made using 2, 4, 8 and 16 groups in turn. For each group, the number of threads used was varied between 1, 2, 4, 8 and 16. For each of the combinations, time taken to schedule and the makespan for each variation was recorded.

Experiment 5 - ETB 2 (varied groups from 2, 4, 8 to 16)

The fifth experiment used the ***ETB*** method to group jobs and the ***EvenDist*** method used to group the machines before implementing the MinMin scheduling algorithm to schedule same range of jobs in experiment 1 between paired groups of jobs and machines. The experiment was executed using 2, 4, 8 and 16 groups in combination with 1, 2, 4, 8 and 16 threads. For each of the combinations, the time taken to schedule and the makespan for each variation was recorded.

Experiment 6 – ETSB1 (varied groups from 2, 4, 8 to 16)

The sixth experiment used the ***ETSB*** method to group jobs and the ***SimTog (SimTog)***method was used to group machines before implementing the MinMin scheduling algorithm to schedule the same range of jobs in experiment 1 above between paired groups of jobs and machines. The experiment was executed with 2, 4, 8 and 16 groups in combination with 1, 2, 4, 8 and 16 threads. For each of the combinations, the time taken to schedule and the makespan for each variation was recorded.

Experiment 7 - ETSB 2 (varied groups from 2, 4, 8 to 16)

The seventh experiment used the ***ETSB*** method to group the jobs and the ***EvenDist*** method to group the machines before implementing the MinMin scheduling algorithm to schedule the same range of jobs in experiment 1 above between paired groups of jobs and machines. Several runs of the experiment were made using 2, 4, 8 and 16 groups in turn. For each group, the number of threads used was varied between 1, 2, 4, 8 and 16. For each of the combinations, the time taken to schedule and the makespan for each variation was recorded.

4.7.2 Relationship between a job, a thread and a group

The GPMS uses two different groups: job group and machine group.

A job group is made up of several jobs (sorted) based on the jobs attributes and the method used for sorting them. Likewise, a machine group is made up of several machines (sorted) based on their configurations and the method used in categorising them. Hence, the relationship between a group and a job is one-to-many.

Threads are lightweight processes or units of execution and multicore systems possess the capacity to concurrently execute processes and threads. Threads are exploited in the GPMS to enhance parallelism and increase scheduling throughput.

Based on the experiments carried out for this research; the number of groups was varied between 2, 4, 8 and 16. Threads were varied from 1 to 16 in steps of power 2 (2^n) to simultaneously execute the scheduling algorithm in parallel. For instance, with two groups, a range of threads from 1 to 16 in steps of power 2 were used. With four groups, a range of threads from 1 to 16 in steps of power 2 were used. With eight groups, a range of threads from 1 to 16 in steps of power 2 were used and so forth. In a typical scenario, the systems administrator would set the number of threads to be used based on performance requirements and system load. However, the GPMS system does not currently have direct control over assigning threads to particular cores or functions. Use of multiple threads though encourages parallelism in the processing.

In this research, results are presented where the number of threads equals the number of groups.

4.7.3 The Grouping of Jobs and Machines in GPMS

The GPMS system employs both job groups and machine groups; jobs are batched before grouping. The system creates the same number of job groups and machine groups by grouping the machines and then for each execution groups the jobs according to the specified number of machine groups. Machine groups and job groups are created and paired before scheduling; hence the number of job groups and machine groups are always equal. It is currently part of the GPMS algorithm to create the same number of job groups and machine groups. The GPMS system therefore is the agent that creates the groups in the first place so can ensure equality. Typically the machines are grouped or re-grouped less frequently than jobs. For instance a grouping of machines would be made (assume N groups) and this grouping would persist until the administrator determined that it was no longer the required grouping. The machine grouping might last days or months. Meanwhile jobs as they enter are batched and when a certain number of jobs have been entered, they are grouped into N groups to match the number of machine groups. Thus job groups and machine groups are always the same and this is part of the GPMS method. However, with modification, the system could adequately respond to situations with differing job and machine groups. For instance, a 'multiple-group pairing' strategy could be implemented to pair more groups of machines to groups of job or vice versa. Multiple-group pairing in this case might involve the pairing of more than one job group to one machine group or pair more than one machine group to one job group. However such multiple-group pairing was not explored in this research since a basic tenet of the GPMS matching is to ensure equal number of groups for jobs and machines.

4.7.4 Combination of the Number of Experiments

In each of the experiments, the MinMin algorithm was executed on an HPC system to schedule a range of jobs from 1000 jobs to 10000 jobs (in steps of 1000). The experiment was controlled in steps of 1000 so that the effect of increasing jobs on the speedup could be determined. In the experimentation the threads were varied from 1 to 16 in power 2 (2^n) and the groups were varied between 2, 4, 8 and 16. Steps of power 2 was considered because

multicore computers exist in that order and a relationship can be easily establish between the number of groups used, number of threads used and number of CPUs used.

The complete experimentation yielded many results because of the combinations of several variables (number of groups, number of threads, job grouping method and machine grouping methods). For each instance of the experiment, the ***timeofScheduling*** (the time taken to do the scheduling) for each set of jobs, for each method, for each number of groups, and for each number of threads was recorded. This combines to give very high number of experiments and results.

For the base experiment (Ordinary MinMin) there were10 scheduling instances(1000 to 10000 in steps of 1000), combined with five possible threads (1, 2, 4, 8, 16), combined with just 1 group number (Ordinary MinMin does not use grouping so one can consider the input job set to be a single group) and combined with the two machine grouping methods.

For the Priority method there were10 scheduling instances (1000 to 10000 in steps of 1000), combined with five possible threads (1, 2, 4, 8, 16), combined with just 1 group number (the Priority method always used 4 groups) and combined with the two machine grouping methods.

For the ETB and ETSB methods, there were of 10 scheduling instances(1000 to 10000 steps 1000), combined with five possible threads (1, 2, 4, 8, 16), combined with four possible group number variations (2, 4, 8, and 16) and combined with the two machine grouping methods.

4.8 Shortcomings of the Grid Workload Archive

In the Grid workload archive, more than 90% of the fields for ReqNProcs have 1 as the value. This attribute (ReqNProcs) was used to determine the priority of the job in the priority grouping method (jobs with ReqNProcs = 1 are sorted to ‘low priority’ group). This impacted heavily the result of experiment on the Priority job grouping method as more of the jobs were sorted to a single group instead of spreading into all four groups. Hence, as the number of jobs increased, the performance of the method decreased against the MinMin. This also informed the decision to implement the ETB and ETSB methods.

Also, in the Grid Workload archive, more than 90% of the values for AverageCPUTimeUsed are not provided and are denoted with -1. The averageCPUTimeUsed may represent the actual execution time of the job on the system as this value in most cases should be from the system after job execution but unfortunately, the values are not provided. This however does not affect the result much because AverageCPUTimeUsed is a second option used only when ReqTime and ReqNProcs are not provided. Furthermore, the sizes of jobs are not provided in the workload. Hence the GPMS system uses attributes (ReqTime and ReqNProcs) in the Grid workload archive to estimate the job sizes and uses the size of job to estimate execution time.

These factors combine to make the Grid workload archive inadequate in its raw form for experiments involving the size of jobs, estimation of processing time and equal spread of requested number of processors. The inadequacy was overcome by the method described previously for estimating job size.

4.9 Summary

This chapter has discussed the design of the Group-based parallel Multi-Scheduler (GPMS) for Grid, the simulations and the various experiments carried out. It started by defining the functions of the system, identifying the components to perform the functions and then described the design of the system. The chapter then described the simulation of Grid site and Grid machines and their attributes. It also described the various experiments and the number of combinations of the experiments or results.

The GPMS is focussed on grouping jobs and machines and then running parallel instances of the MinMin scheduling algorithm within paired job-machine groups. Various grouping methods have been developed and the design of these was presented in this chapter.

The next chapter presents the results of the experimentation and the analysis of the GPMS methods.

CHAPTER FIVE

RESULTS AND ANALYSIS OF THE GPMS METHODS

CHAPTER FIVE
RESULTS AND ANALYSIS OF THE GPMS METHODS

5.1 Introduction

This chapter presents the results from the experiments discussed in Chapter Four, and the evaluation of each of the GPMS methods results against the ordinary MinMin. The analysis and evaluation is presented in four sections in which results of the Priority method, the ETB method and the ETSB method respectively are compared against the ordinary MinMin. This is followed by a comparative analysis of all the GPMS methods.

5.2 Results and Performance Evaluation of the Priority Method

This section discusses results and analysis of experiment 2 and experiment 3 and evaluation of the Priority method against the ordinary MinMin (experiment 1). Comparison was also made between the two machine grouping methods to ascertain which one works better with the Priority method.

5.2.1 Presentation of Results (Priority)

The analysis of the results presented in this section is based on four threads in order to create a one-to-one relationship between groups and threads. Also, the other results exhibited the same pattern or characteristic across threads. Four threads was chosen for presentation because this represents the median of threads used and also because it easily matched the four groups used in the Priority method. Two methods of evaluation were used. These were termed speedup and performance improvement. The speedup was evaluated against the ordinary MinMin at each scheduling interval but performance improvement was evaluated against the MinMin and also between the successive groups using the total scheduling time.

The Priority method splits jobs into priority groups based on their attributes. Machines are also split into the same number of groups based on their configurations. Each priority group of jobs is then paired to a machine group before the MinMin scheduling algorithm is executed within groups in parallel. A priority group in this context means a group containing

a collection of jobs which have been determined to have a similar priority. These jobs might possess some similar characteristics or meet certain requirements that resulted in them being sorted into the same priority group.

The result for the experiments and the computation of correlation, Analysis of Variance (ANOVA) significance test and standard deviation is shown in Table 20. Table21 shows the computation of improvement in multiples and in percentages. The correlation results between the methods shows a general pattern across methods – that results are strongly correlated with values close to 1 (0.9x). The ANOVA significance test also shows a significant difference between the Priority methods and the MinMin. The standard deviation and mean of the MinMin algorithm was **19831.78** and **24203.3** respectively). This means that the result of the MinMin is scattered from its mean. The standard deviation and mean of the PrioritySimTog method was **4085.54** and **4100.6** respectively. The standard deviation and mean of the PriorityEvenDist was **3845.52** and **3580.7** respectively. This means that the results for the Priority methods are spread closer to the mean. See Table 20 for values of standard deviation.

Table 22 shows the computed speedup against the MinMin algorithm as the scheduling continues from 1000 jobs to 10000 jobs. The raw results show differences in scheduling time between the two machine grouping methods (EvenDist and SimTog). However, the Anova computation shows no significant difference between results of the two machine grouping methods. Figures 13, 14, 15, 16, 17and 18 illustrate how the Priority method compares with the MinMin method.

Figure 13 shows the percentage average and total scheduling times used by the methods (ordinary MinMin, Priority-EvenDist and Priority-SimTog) to schedule the same range of jobs. It shows that the ordinary MinMin took a total of 242033 Milliseconds, the SimTog method used 41006 Milliseconds and the EvenDist method used 35807 Milliseconds to schedule same range of jobs. In percentage, the MinMin used 76% of the time to schedule the jobs, while using the Priority method, the SimTog method used 13% of the time to schedule the same jobs and the EvenDist method used just 11% of the total time to schedule the same range of jobs.

The EvenDist method recorded between 5.0 to 11.8 times speedup (with an average of 6.8 times) speedup against the ordinary MinMin algorithm while the SimTog method recorded 5.0 to 9.6 times (with an average of 5.9 times) speedup against the ordinary MinMin

algorithm as the number of jobs increases from 1000 to 10000 (see Table 22 and Figure 14). The best speedup is achieved when number of jobs equals 4,000. This may be due to jobs at this stage being more balanced into the four groups. At this point, the speedup was equal to 11.8 for the EvenDist method and 5.9 for the SimTog method.

The EvenDist method recorded from 80% to 92% speedup with an average of 87% speedup over the ordinary MinMin algorithm and the SimTog method recorded a range of 80% to 90% with an average of 85% speedup against the ordinary MinMin algorithm (see Figure 15).

Figures 14 and Figure 15 show the speedup in multiples and speedup in percentage by the Priority method over the ordinary MinMin. The speedup improved from 6.9 times to a maximum of 11.8 times as the number of jobs increased from 1000 jobs to 4000 jobs. Then it began a downward trend. This negative slope of the speedup as the number of jobs increases indicates that even though the method was generally better than the MinMin, performance was degrading as the number of jobs increased. This is attributable to the type of jobs used in the experiment and the Priority method for grouping jobs and is discussed in section 6.2.2.

Figure 16 shows the time used by the methods to schedule as the number of jobs increases. It shows that as the number of jobs increases from 1000 to 10000, the scheduling time also increases but the scheduling time of the MinMin increases faster. This is because the Priority method distributes the jobs into groups before scheduling the jobs in parallel. Figure 17 also compares the total and average scheduling times of the methods used. It shows that the Priority grouping methods performed better than the ordinary MinMin. Figure 18 shows the performance chart for the MinMin and the Priority methods and also shows the polynomial nature of the methods. The Priority methods performed far better than the MinMin but the performance was degrading as the number of jobs increases. This was because jobs were not uniformly distributed based on priorities. Hence, more jobs were being sorted to and scheduled from one group.

Table 20 Results and computation of correlation, ANOVA and standard deviation (Priority)

Jobs Limit	MinMin	Priority Method		Correlation	ANOVA Significance Test	Standard Deviation
		EvenDist	SimTog			
1000	654	95	105	Between MinMin and EvenDist = **0.9740** (Strongly correlated)	Between MinMin and EvenDist P-value=**0.006** (significant)	MinMin = **19831.78** (Less than and wide from mean of 24203.3)
2000	3230	340	412			
3000	7601	673	839			
4000	12920	1092	1345			
5000	18219	1776	2008			
6000	22671	2837	3339	Between MinMin and SimTog =**0.9895** (Strongly correlated	Between MinMin and SimTog P-value=**0.006** (significant)	Priority-EvenDist = **3845.52** (greater and close to the mean of 3580.7)
7000	29504	3860	4570			
8000	39074	5312	7500			
9000	48178	7818	8830			
10000	59982	12004	12058			
Total	**242033**	**35807**	**41006**	Between EvenDist and SimTog = **0.9876**(Strongly correlated)	Between EvenDist and SimTog P-value =**0.772** (not significant)	Priority-SimTog = **4085.54** (Less than and very close to mean of 4100.6)
Ave	**24203.3**	**3580.7**	**4100.6**			

Table 21 Performance in multiples and in percentage

Performance Improvement	**MinMin**	**EvenDist**	**SimTog**
TotalSchedTime	**242033**	**35807**	**41006**
Performance Improvement In Multiples Where $x_1 = Total_{MinMin}$ and $x_2 = Total_{PriorityEvenDist}$ or $x_2 = Total_{PrioritySimTog}$		$\frac{x_1}{x_2} = 6.76$	$\frac{x_1}{x_2} = 5.90$
Performance Improvement In Percentage Where $x_1 = Total_{MinMin}$ and $x_2 = Total_{PriorityEvenDist}$ or $x_2 = Total_{PrioritySimTog}$		$\frac{x_1 - x_2}{x_1} * 100$ $= 85.20574$	$\frac{x_1 - x_2}{x_1} * 100$ $= 83.05768$

Table 22 Speedup in percentage and in multiples

				Speedup (%)		**Speedup (X)**	
	Schedule Time in seconds						
JobsLimit	MinMin	EvenDist	SimTog	EvenDist	SimTog	EvenDist	SimTog
1000	0.7	0.1	0.1	85%	84%	6.9	6.2
2000	3.2	0.3	0.4	89%	87%	9.5	7.8
3000	7.6	0.7	0.8	91%	89%	11.3	9.1
4000	12.9	1.1	1.3	**92%**	**90%**	**11.8**	**9.6**
5000	18.2	1.8	2.0	90%	89%	10.3	9.1
6000	22.7	2.8	3.3	87%	85%	8.0	6.8
7000	29.5	3.9	4.6	87%	85%	7.6	6.5
8000	39.1	5.3	7.5	86%	81%	7.4	5.2
9000	48.2	7.8	8.8	84%	82%	6.2	5.5
10000	60.0	12.0	12.1	**80%**	**80%**	**5.0**	**5.0**
Total	**242.1**	**35.8**	**40.9**	-	-	-	-
Average	-	-	-	**87%**	**85%**	**6.8**	**5.9**

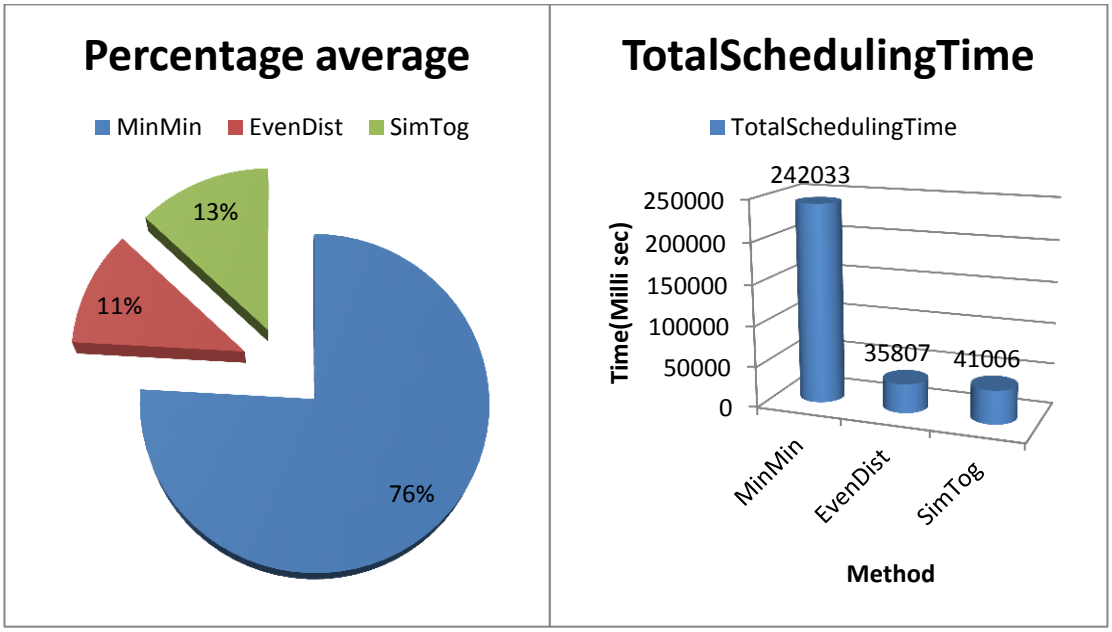


Figure 14: Percentage average and total scheduling times for MinMin and Priority

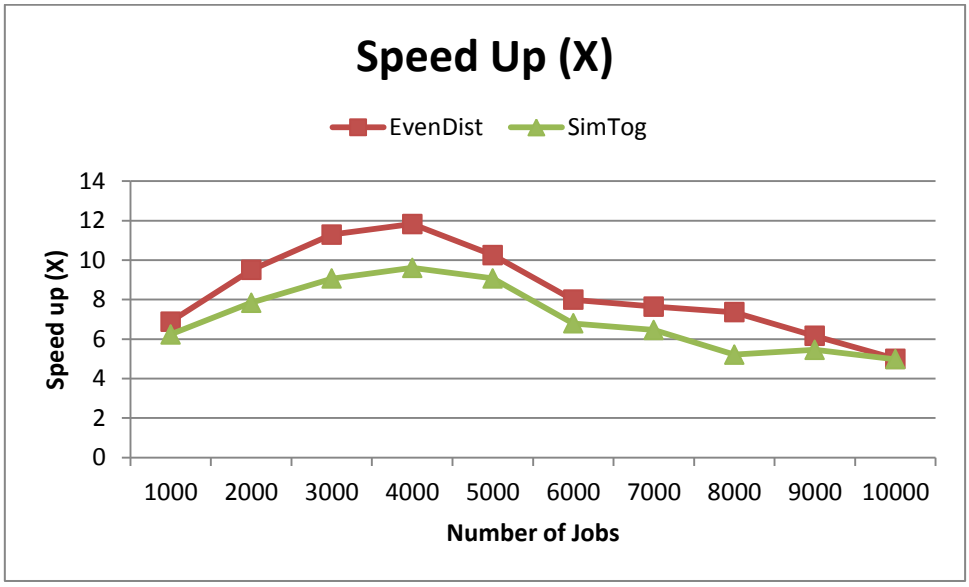


Figure 15: Speedup in multiples by Priority over MinMin

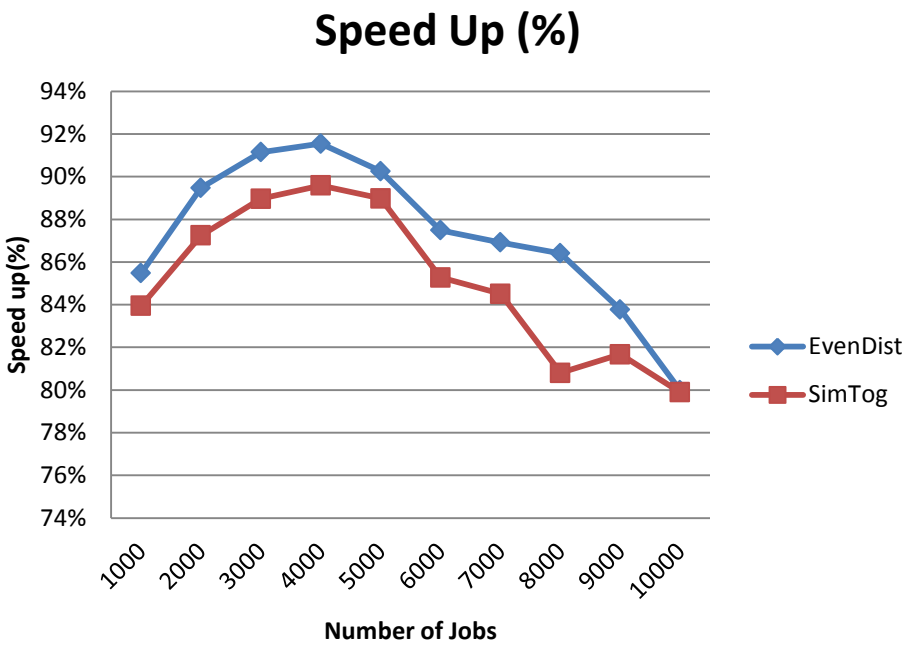


Figure 16: Speedup in percentage by Priority over MinMin

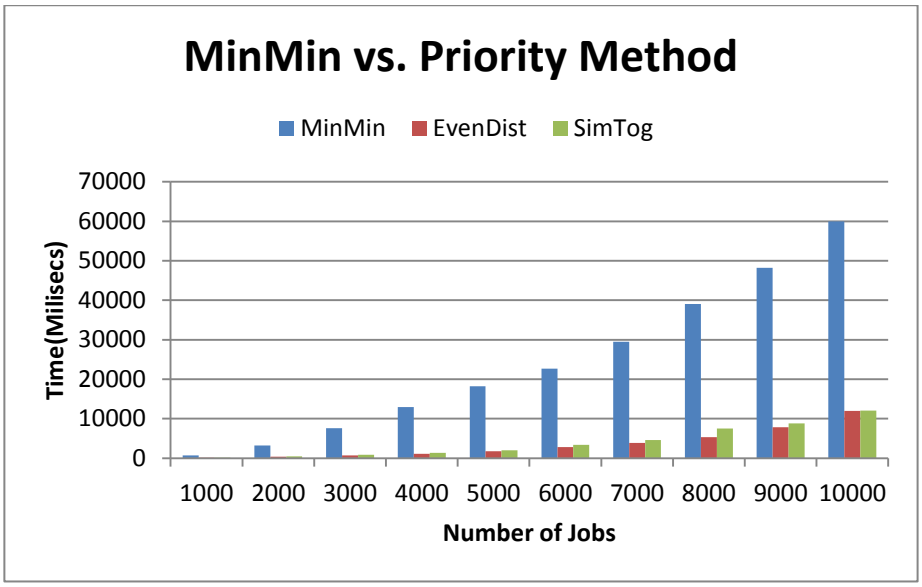


Figure 17: Total scheduling time of Priority and MinMin with increasing number of jobs

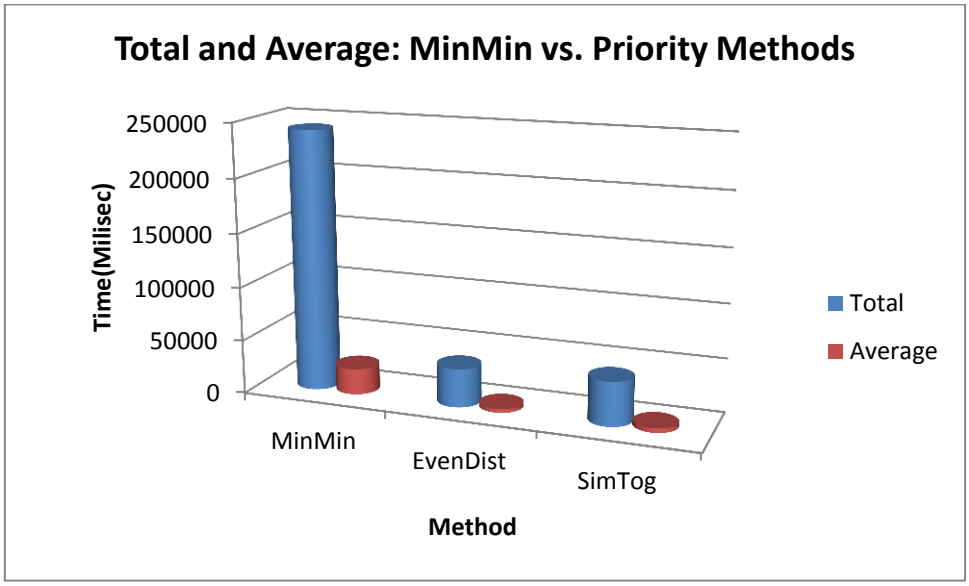


Figure 18: Total and average scheduling time of Priority and MinMin

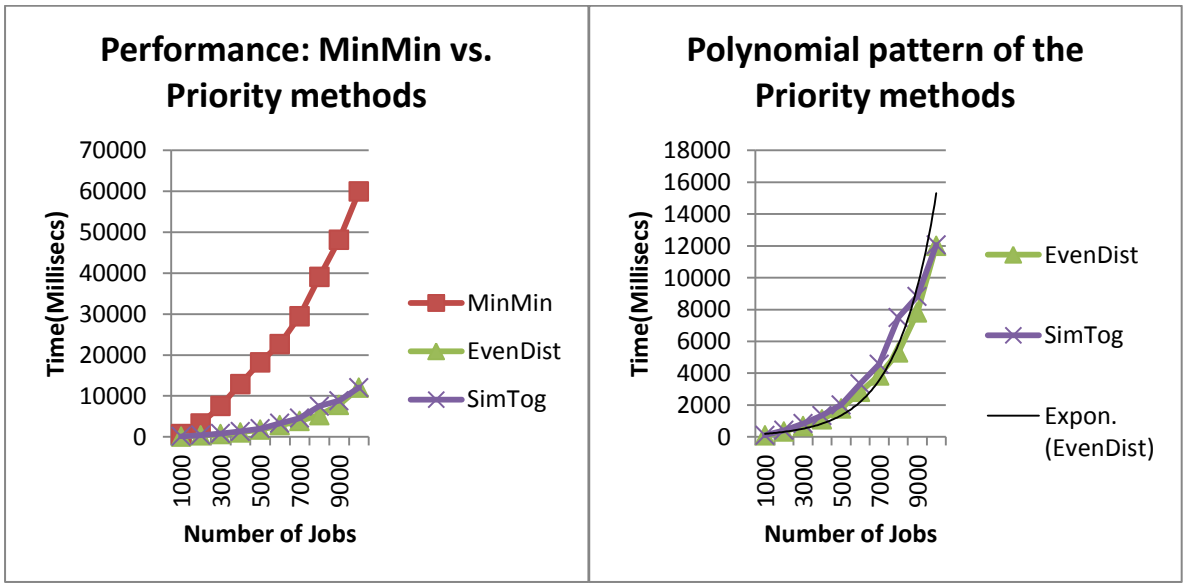


Figure 19: Polynomial pattern of the Priority methods

5.2.2 Discussion of Results (Priority)

There was a significant difference in performance between the Priority method results and that of the MinMin. The EvenDist method performed better than the MinMin by 6.76 times representing 85% while the SimTog method performed better than the MinMin by 5.9 times representing 83%. The EvenDist method also performed better than the SimTog method by some margins but the difference was not significant from the ANOVA test carried out. Figure 13, Figure 14, Figure 15, Figure 16 and Figure 17 shows the graphs detailing the performance of the three methods, while Table 20 shows the analysis of variance, correlation and standard deviation.

Though both EvenDist and SimTog methods performed better than the MinMin algorithm, the pattern of the graph for both methods was generally polynomial (see Figure 18). The performance was degrading relatively as the number of jobs increases. This is exacerbated by the fact that more jobs in the test data set were scheduled to one machine group while the other machine groups ended up with fewer jobs. Hence, as the number of jobs increases, the number of jobs in that one priority group approaches same number of jobs as in the non-grouping method, thereby degrading the general performance.

This effect can be dampened by making sure that jobs are equally distributed among the groups. Polynomial time has the characteristic that as the number of instances of the input set increases so does the time per instance. Thus grouping jobs to create smaller sets and scheduling in parallel improves performance. Smaller sets and parallel execution/scheduling reduce the time required per instance as the total time required to schedule each set is greatly reduced.

With the Priority method, it cannot be guaranteed that the jobs are equitably distributed among the groups. There is always the possibility that while some groups are still very busy scheduling jobs, others will have finished scheduling and remain idle. In the experiment, this affected the overall schedule time. This observation prompted the researcher to explore methods that can at least ensure that jobs are to a large extent equitably distributed among the groups.

The Priority method aims at improving scheduling time. However improvements in

scheduling time are not valuable if the resulting schedule is inferior to one which would have been produced via a slower scheduling algorithm. Where many of the input jobs have the same priority, they will be assigned to the same group of machines which could cause an imbalance in processing activity. This might result in a poorer schedule than would have been the case if machines were not grouped. On the other hand, if the priority is evenly distributed across the input jobs, then the resulting schedule is likely to be equal to or of better quality to one produced without grouping. If the machines are normally distributed and the EvenDist method is used for machine grouping, then the execution time should be the same for all priority groups, whereas if SimTog is used the execution time and quality of service could be improved if higher priority and larger jobs could be assigned to the groups with better machine configuration. In these cases, makespan should be improved, where makespan is considered to be a combination of both scheduling time and execution time. However much depends on the exact requirements of the incoming jobs and the characteristics of the receiving Grid. Thus it can be concluded that an even balance of jobs across groups is desirable and also that tuning the scheduling parameters according to incoming job characteristics would be beneficial towards achieving a better schedule.

Since there were only four groups used in the experiment, the effect of grouping on performance cannot be fully ascertained. It would be worthwhile to implement a method where the number of groups is not restricted by the method itself. Effort should focus on methods that allow the number of groups to be varied just as the number of threads. This will throw more light on the effect of grouping on the performance of scheduling algorithms. Also, the impact of performance, grouping and number of processors used for execution needs to be explored further.

Another observation worth mention and discussion is how both machine grouping methods obtained the highest speedup against the MinMin at the point when the number of jobs equals 4000. The SimTog method recorded 9.6 as highest speedup and the EvenDist method recorded 11.8 as highest speedup – all at the point when number of jobs equal 4000. Could it be that the four thousand jobs were better shared into the four priority groups and scheduled more effectively? This phenomenon also calls for further investigation for a method that distributes jobs equally into groups despite their attributes to enhance scheduling. Following these results, the researcher investigated other grouping methods.

Employing the Priority method improves the performance of the MinMin scheduling algorithm substantially but because the number of groups was constant, the relationship between varying (increasing) the number of groups and improvement in scheduling efficiency cannot be ascertained. More of the jobs were allocated to a single priority group as they exhibited similar characteristics – this also affected the performance of the scheduler as the number of jobs increased. Hence, further investigation is required for methods that ensure jobs are equally distributed into groups and number of groups is variable. This will reveal the effect of increasing the number of groups on scheduling efficiency. The design of the ETB and ETSB were therefore proposed. These two methods are intended to correct the shortcomings inherent in the Priority method.

5.3 Results, Analysis and Evaluation of the ETB Method

5.3.1 Presentation of Results (ETB)

This section presents results and analysis of experiment 4 and experiment 5 which comprise the evaluation of the ETB method against the ordinary MinMin (experiment 1).

The ETB method seeks to improve on some of the drawbacks inherent in the Priority method. Hence, it uses a method that ensures jobs are evenly or equally spread into groups. The method uses an estimation of the processing time (or execution time) for each job to group the jobs. It attempts to share jobs equally into all groups by trying to even out the total processing times or execution time of jobs in all groups. It does this by selecting a job and adding it to a group with the least total execution time. For each job added to a group, the **totalestimatedTime** for the group is updated by adding the execution time of the job to that of the group. Then the next job is selected and the group with the least total execution is picked as the candidate for addition. By adding the next job to the group with the current lowest total processing time, the method ensures that jobs are spread equally into all groups – even if not by number. Machines are distributed into same number of groups as jobs – this is to enable a one-to-one pairing between job groups and machine groups. Pairings are then made between job groups and machine groups, and then multiple instances of the MinMin scheduling algorithm are executed within paired groups (multi-scheduling) using multiple threads (multithreading) in parallel.

Table 23 and Table24 show the results and computation of speedup over the MinMin algorithm by the ETB methods using 2 to 8 groups. The MinMin used a total of 242033ms and an average of 24203.3 ms to schedule the job sets. Using two groups, the ETB-EvenDist method used a total of 34862ms and an average of 3486.2 ms to schedule the same range of tasks. Four groups used a total of 4701 ms and an average of 470.1 ms to schedule the task, while eight groups used a total of 1435ms and an average of 143.5 ms to schedule same tasks.

In the same vein, the ETB-SimTog used a total of 34667ms and an average of 3466.7ms to schedule the same tasks when using 2 groups. It used a total of 5224ms and an average of 522.4ms for 4 groups to schedule same tasks and a total of 1541ms and an average of 154.1ms for 8 groups to schedule the same set of jobs.

Figure19 shows the total scheduling time and average scheduling time for ETB-EvenDist. It shows that with the ETB-EvenDist, the ordinary MinMin took 86% of the total time, 2 groups took 12% of the time, 4 groups used just 2% of the time to schedule same range of tasks while the time used by 8 groups is very negligible compared to the rest. Figure 20 shows total scheduling time and average scheduling time for the ETB-SimTog method.

From Table 23 and Figure 21, the ETB-EvenDist method exhibited similar pattern across all groups (2, 4 and 8). The speedup increased to a point then declines as the number of jobs increases from 1000 to 10000. For instance, using 2 groups, the speedup in multiples improved from 6.48 times (at 1000 jobs) to 9.92 times (at 3000 jobs). The speedup then declines as the number of jobs increases to 10000. With 4 groups, the speedup improved from 16.35 (at 1000 jobs) to 59.19 times (at 6000 jobs) before declining while using 8 groups, the performance improved from 59.45 times (at 1000 jobs) to 182.19 times (at 5000 jobs). The performance then declines as the number of jobs increases to 10000.

These results represent a significant performance improvement over the MinMin algorithm on group basis. For instance, when scheduling with two groups, the ETB-EvenDist and ETB-SimTog recorded an average of 6.94 and 6.98 times performance improvement over the MinMin respectively. Using four groups, the performance improvement was 51.49 and 46.33 times respectively over the MinMin. When using eight groups, the performance improved over the MinMin by 168.66 and 157.06 times respectively. Table 25 provides the ANOVA test results which reveal the significance differences between the groups. Taking P values less than 0.05 to indicate significance, the analysis showed that all differences were found to be

highly significant with very low P values. For instance, there were significant differences between the MinMin and the ETB-EvenDist and between the MinMin and the ETB-SimTog methods. Significant differences were also found between successive ETB groups. This meant that increasing the number of groups impacted the result considerably.

Table 23 shows the result of experiments for the ETB-EvenDist method and the speed in multiples and in percentage. Table24 shows the speedup in multiples and in percentage for the ETB-SimTog method. Significant speedup was recorded at each level of job scheduling, scheduling from 1000 to 10000 jobs in steps of 1000. Using 2 groups, the ETB-EvenDist method recorded between 6.32 to 9.92 times speedup with an average of 7.62 times speedup against the MinMin. Using four groups, the ETB-EvenDist method recorded between 16.35 to 59.19 times with an average of 47.46 times speedup over the MinMin. Eight groups recorded between 59.45 and 182.50 times speedup and an average of 155.33 times speedup over the MinMin. In the same vein, when using two groups, the ETB-SimTog recorded between 5.33 to 11.10 times speedup with an average of 8.15 times speedup against the MinMin. Using four groups, the method recorded between 20.44 to 76.78 range speedup and an average of 50.39 times speedup against the MinMin. And with 8 groups, the method recorded between 65.40 to 187.82 range of speedup and an average of 147.28 times speedup against the MinMin. Across all the groups, as the number of jobs increases, there was a general improvement in the speedup to a point beyond which the rate of speedup declines. Figure 21 and Figure 22 shows the speedup in multiple while Figure 23 and Figure 24 show the speedup in percentage for the ETB-methods.

There was a significant performance improvement by the ETB methods over the MinMin as the number of groups increased. Increasing the number of groups decreases the number of jobs per group and therefore decreases the total scheduling time. Figures 25, 26 and 27 show the scale of the improvement recorded against the MinMin by the ETB methods with increasing number of groups. As the number of groups changes from two groups to eight groups, the scheduling efficiency improved significantly over the MinMin. This shows that using more groups increases the performance of the scheduling algorithm. Figure 25 shows that as the number of groups changes between 2, 4 and 8, the ETB-EvenDist method recorded 6.94 times, 51.49 times and 168.66times improvements respectively. While for the ETB-SimTog, the improvements recorded by 2, 4 and 8 groups were 6.98, 46.33 and 157.06 times respectively.

The GPMS sorts a number of jobs into independent groups from where scheduling operations can take place in parallel. The number of groups used range between 2 and 16 groups. For the method to achieve high scheduling efficiency against other scheduling algorithm, it is required that each group has a number of jobs to schedule in parallel. Hence, if the number of jobs to be scheduled is low or equal to the number of groups, the experiment can be set up but the GPMS method might not record significant gain over the other scheduling algorithms. This is so because of overheads in making and maintaining groups outweigh the advantages of group parallel scheduling when the number of jobs is low or equal to the number of machine groups.

Figure 26 shows the improvement of the ETB-EvenDist method over MinMin and between successive groups. Figure 27 shows the improvement of the ETB-SimTog method over MinMin and between successive groups. Although there was a general performance improvement over the MinMin as the number of groups increases, the rate of performance improvement of a successive group over its predecessor (within same method) decreases generally**.** For instance, using the ETB and EvenDist method, the rate of improvement of two groups over the ordinary MinMin was 6.94. As the group increased from 2 groups to 4 groups, there was performance improvement of 47.46 over the MinMin but between 2 groups and 4 groups within same method, the improvement rate was just 7.41. Furthermore, as the group increased from 4 groups to 8 groups, performance of the method over the MinMin improved 155 times but between 8 groups and 4 groups, the improvement was only 3.28 times and 8 groups performed better than 2 groups by 24.29 times. This slowdown in performance between successful groups is caused by shared rersurces contention between increased threads.

Figure 27 shows the improvement of the ETB-SimTog method over MinMin and between successive groups. Using the ETB-SimTog method, 2 groups improved about 6.98 times over the MinMin and 4 groups showed improvement of 46.33 times over the MinMin but between2 groups and 4 groups, performance improved by just 6.64 times. Moving from 4 groups to 8 groups, there was performance improvement of 157.06times over the MinMin but 8 groups performed better than 4 groups by just 3.39 times. In the same vein, 8 groups performed better than 2 groups by about 22.50 times.

This shows that even though there is a general performance improvement over MinMin with increasing groups, the performance does not continue to improve at the same rate with increasing group within the method due to performance limiting factors. This is attributable partially to the increased number of threads necessitated by successive groups. Increase in threads results in increase resource contention among the threads and this impacted on the result.

The decreasing rate of improvement with increasing groups for ETB-EvenDist and ETB-SimTog is shown in Table 26 and Table 27 and Figures 28 and 29 respectively.

Table 23 Result and speedup for MinMin and ETB-EvenDist

Methods	**MinMin vs. ETB-EvenDist Time in ms**				**Speedup (X) in multiples**			**Speedup (%) in percentage**		
Jobs Limit	**MinMin**	**2Grps**	**4Grps**	**8Grps**	**2Grps**	**4Grps**	**8Grps**	**2Grps**	**4Grps**	**8Grps**
1000	654	101	40	11	6.48	16.35	59.45	84.56	93.88	98.32
2000	3230	331	92	25	9.76	35.11	129.20	89.75	97.15	99.23
3000	7601	766	163	46	**9.92**	46.63	165.24	**89.92**	97.86	99.39
4000	12920	1475	252	76	8.76	51.27	170.00	88.58	98.05	99.41
5000	18219	2410	323	100	7.56	56.41	**182.19**	86.77	98.23	**99.45**
6000	22671	3211	383	128	7.06	**59.19**	177.12	85.84	**98.31**	99.44
7000	29504	4670	511	185	6.32	57.74	159.48	84.17	98.27	99.37
8000	39074	5565	729	228	7.02	53.60	171.38	85.76	98.13	99.42
9000	48178	6989	954	294	6.89	50.50	163.87	85.49	98.02	99.39
10000	59982	9344	1254	342	6.42	47.83	175.39	84.42	97.91	99.43
Total	**242033**	**34862**	**4701**	**1435**	**76.19**	**474.63**	**1553.32**	**865.27**	**975.80**	**992.84**
Average	**24203.3**	**3486.2**	**470.1**	**143.5**	**7.62**	**47.46**	**155.33**	**86.53**	**97.58**	**99.28**

Table 24 Results and speedup for MinMin and ETB-SimTog

Methods	MinMin vs. ETB-SimTog Time in ms				Speedup (X) in multiples			Speedup (%) in percentage		
Jobs Limit	MinMin	2Grps	4Grps	8Grps	2Grps	4Grps	8Grps	2Grps	4Grps	8Grps
1000	654	102	32	10	6.41	20.44	65.40	84.40	95.11	98.47
2000	3230	371	50	28	8.71	64.60	115.36	88.51	98.45	99.13
3000	7601	745	99	46	10.20	**76.78**	165.24	90.20	**98.70**	99.39
4000	12920	1164	196	70	**11.10**	65.92	184.57	**90.99**	98.48	99.46
5000	18219	1860	324	97	9.80	56.23	**187.82**	89.79	98.22	**99.47**
6000	22671	2678	522	173	8.47	43.43	131.05	88.19	97.70	99.24
7000	29504	4046	703	221	7.29	41.97	133.50	86.29	97.62	99.25
8000	39074	5181	907	282	7.54	43.08	138.56	86.74	97.68	99.28
9000	48178	7267	992	288	6.63	48.57	167.28	84.92	97.94	99.40
10000	59982	11253	1399	326	5.33	42.87	183.99	81.24	97.67	99.46
Total	**242033**	**34667**	**5224**	**1541**	**81.48**	**503.89**	**1472.78**	**871.27**	**977.56**	**992.55**
Average	**24203.3**	**3466.7**	**522.4**	**154.1**	**8.15**	**50.39**	**147.28**	**87.13**	**97.76**	**99.25**

Table 25 ANOVA results for ETB-EvenDist, MinMin and between group cardinality

Test	Method	P-value	Significant Difference?
1	MinMin / ETB-EvenDist (All groups)	0.001995	Yes
2	MinMin/ ETB-EvenDist (2Grps)	0.00431	Yes
3	MinMin/ ETB-EvenDist (4Grps)	0.00136	Yes
4	MinMin/ ETB-EvenDist (8Grps)	0.00121	Yes
5	ETB-EvenDist (2Grps)/ ETB-EvenDist (4Grps)	0.006842	Yes
6	ETB-EvenDist (2Grps)/ ETB-EvenDist (8Grps)	0.003126	Yes
7	ETB-EvenDist (4Grps)/ ETB-EvenDist (8Grps)	0.022274	Yes

Table 26 Performance of ETB-EvenDist against MinMin and between groups

Methods		ETB-EvenDist Performance Improvement(X)			ETB-EvenDist Performance Improvement (%)			
Algorithm	MinMin	2Grps	4Grps	8Grps	MinMin	2Grps	4Grps	8Grps
Total	242033	34862	4701	1435		34862	4701	1435
$\frac{Total_{MinMin}}{Total_{Group}}$ Better than MinMin		**6.94**	**51.49**	**168.66**	$\frac{x_1 - x_2}{x_1} X 100$ x_1 = MinMin	**85.60** x_2 =2Grps	**98.06** x_4 =4Grps	**99.41** x_8 =8Grps
$\frac{Total_{nGrps}}{Total_{n+1Grps}}$ $n \in [2,4,8]$ Better than 2 groups			**7.41**	**24.29**	x_1 = 2Grps		**86.52** x_2 =4Grps	**95.88** x_2 =8Grps
$\frac{Total_{nGrps}}{Total_{n+1Grps}}$ $n \in [2,4,8]$ Better than 4 groups				**3.28**	x_1 = 4Grps			**69.47** x_2 =8Grps

Table 27 Performance of ETB-SimTog against MinMin and between groups

Methods		ETB-SimTog Performance Improvement(X)			ETB-SimTog Performance Improvement (%)			
Algorithm	**MinMin**	**2Grps**	**4Grps**	**8Grps**	**MinMin**	**2Grps**	**4Grps**	**8Grps**
Total	242033	34667	5224	1541		34667	5224	1541
$\frac{Total_{MinMin}}{Total_{Group}}$ Better than MinMin		**6.98**	**46.33**	**157.06**	$\frac{x_1 - x_2}{x_1} X100$ X_1 = MinMin	**85.68** X_2 =2Grps	**97.84** X_2 =4Grps	**99.36** X_2 =8Grps
$\frac{Total_{nGrps}}{Total_{n+1Grps}}$ $n \in [2,4,8]$ Better than 2 groups			**6.64**	**22.50**	X_1 = 2Grps		**84.93** X_2 =4Grps	**95.55** X_2 =8Grps
$\frac{Total_{nGrps}}{Total_{n+1Grps}}$ $n \in [2,4,8]$ Better than 4 groups				**3.39**	X_1 = 4Grps			**70.50** X_2 =8Grps

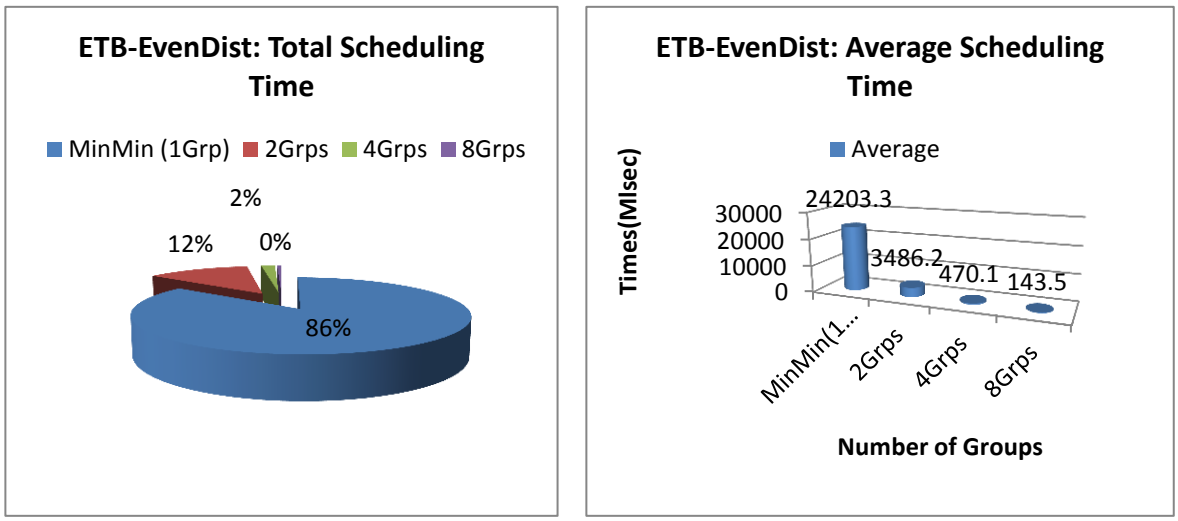


Figure 20: Total and Average scheduling time for ETB-EvenDist and MinMin

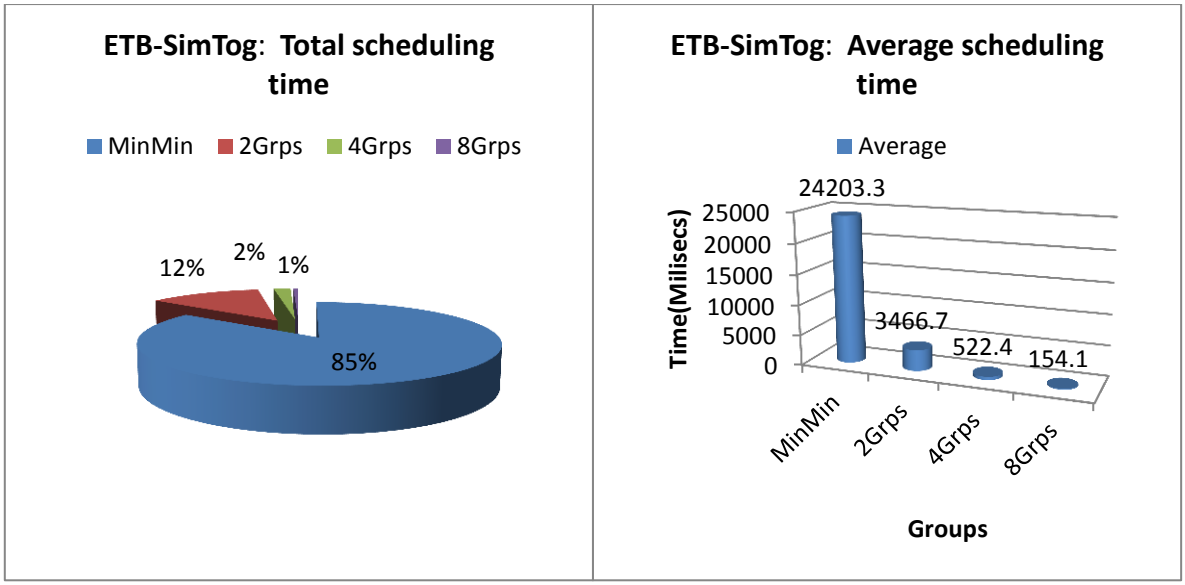


Figure 21: Total and Average of scheduling time for ETB-SimTog and MinMin

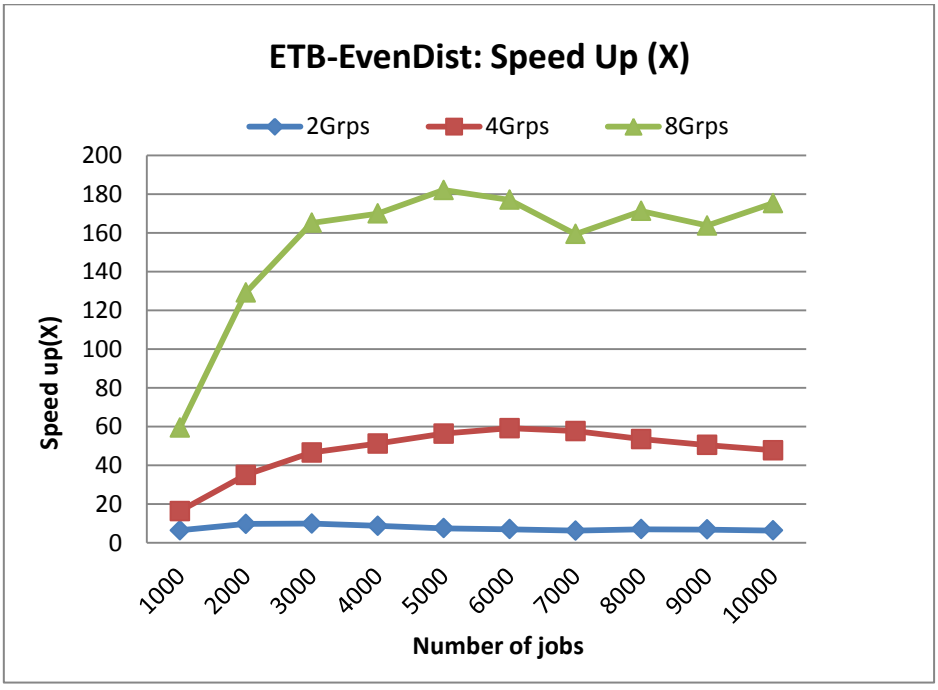


Figure 22: Speedup (in multiples) of the ETB-EvenDist over MinMin

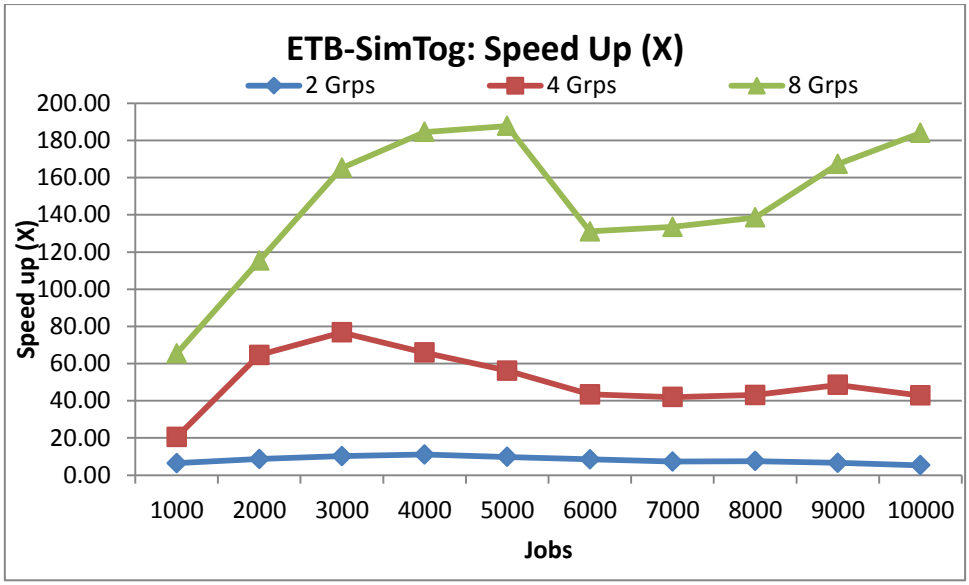


Figure 23: Speedup (in multiples) of the ETB-SimTog over MinMin

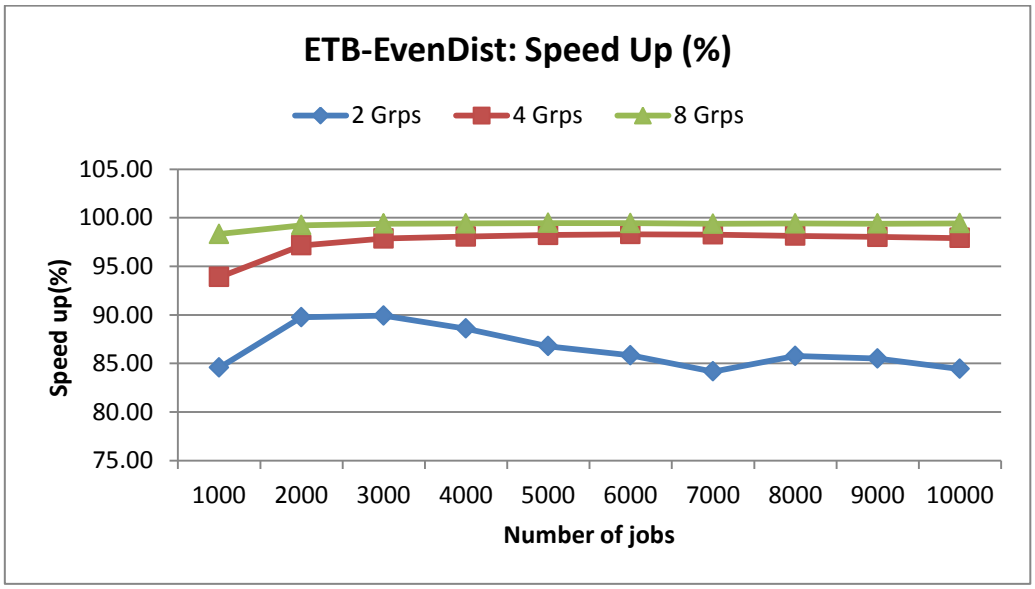


Figure 24: Speedup (in percentage) of the ETB-EvenDist over the MinMin

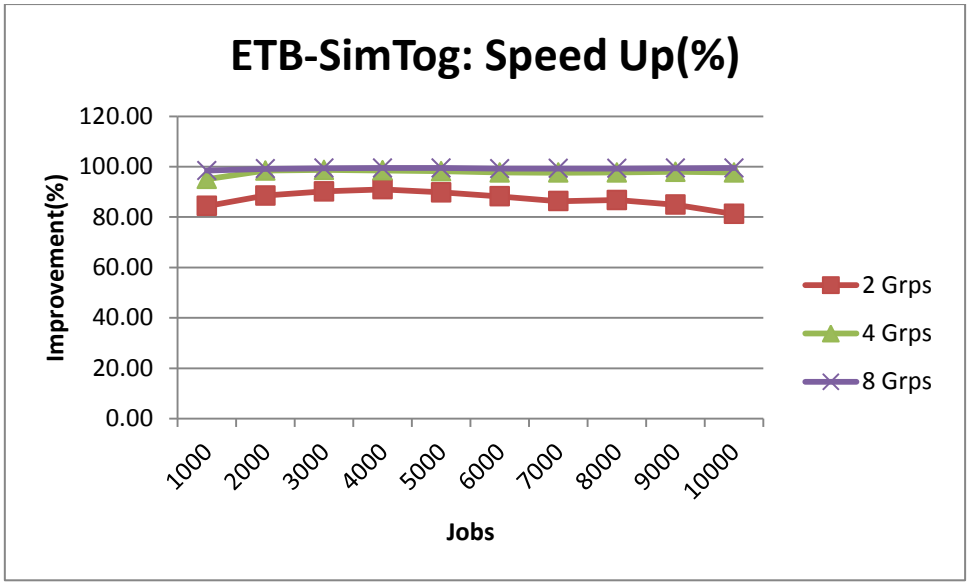


Figure 25: Speedup (in percentage) of the ETB-SimTog over the MinMin

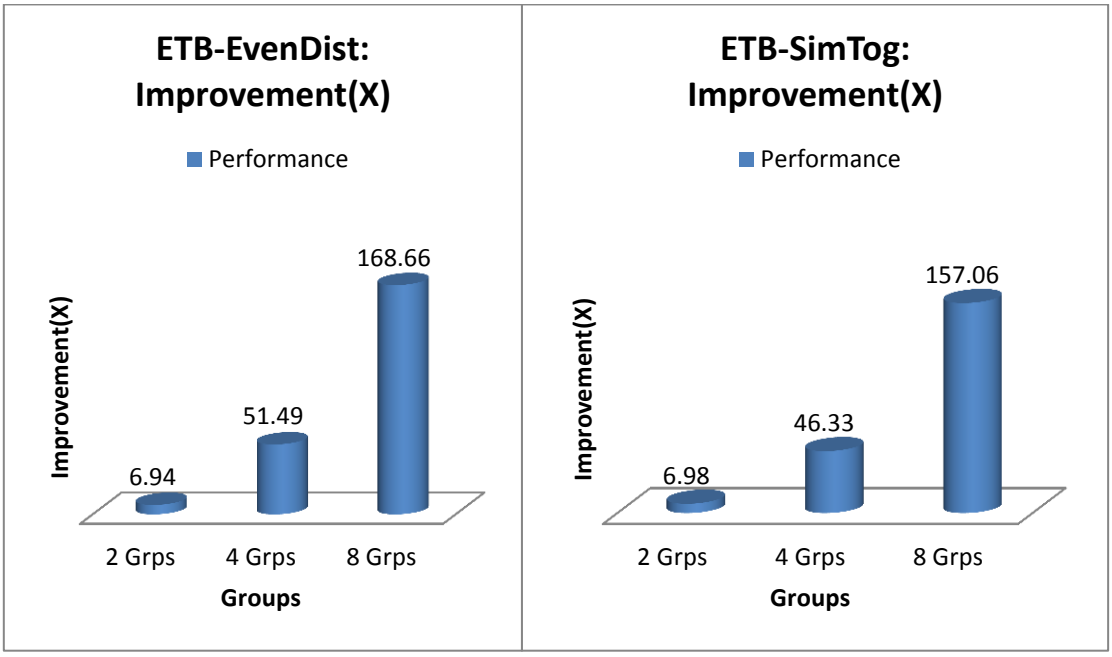


Figure 26: Performance of ETB methods over MinMin across groups

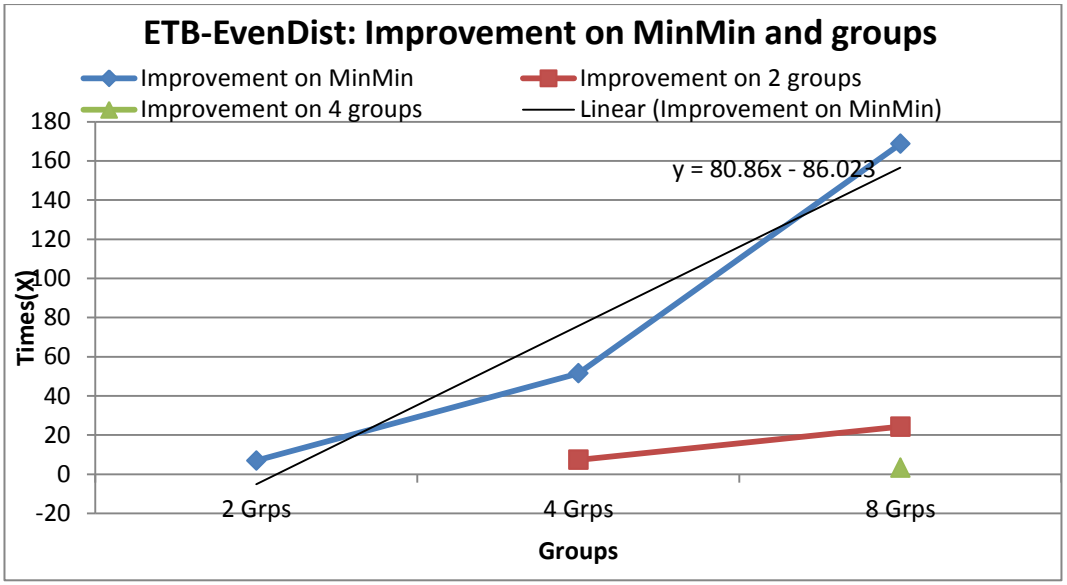


Figure 27: ETB-EvenDist: Improvement on MinMin and across groups

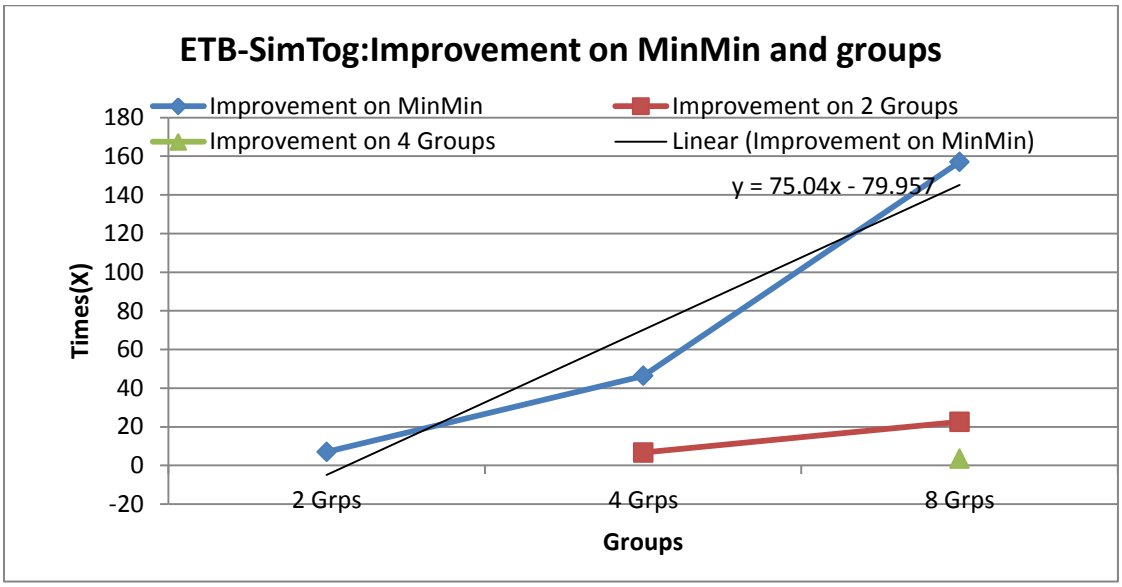


Figure 28: ETB-SimTog: Improvement on MinMin and across Groups

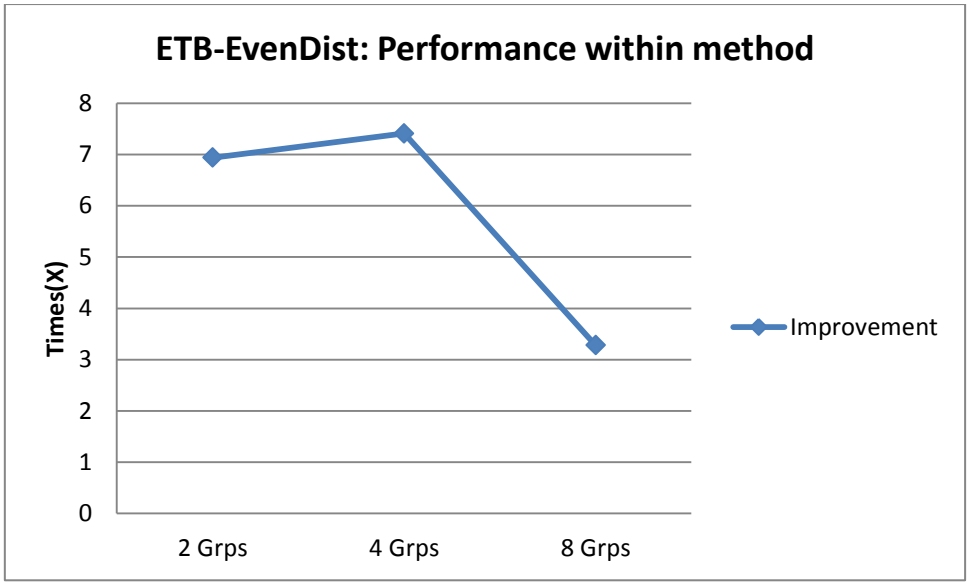


Figure 29: Declining rate of improvement between groups within ETB-EvenDist

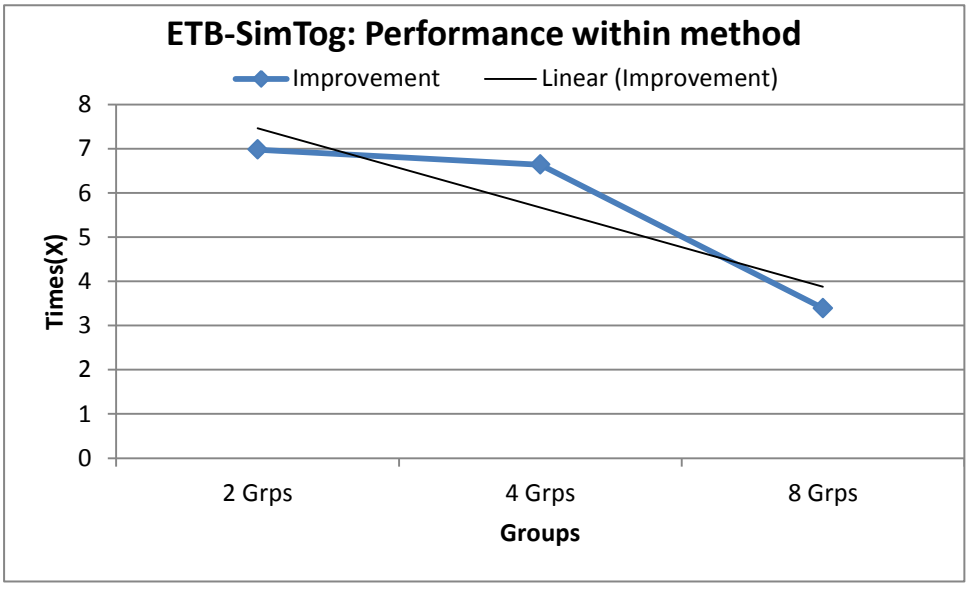


Figure 30: Declining rate of improvement between groups within ETB-SimTog

5.3.2 Discussion of Results (ETB)

Results from the ETB method showed significant performance improvement over the MinMin algorithm. The method allowed the number of groups to be increased between 2, 4 and 8. There was increasing performance improvement over the MinMin as the number of groups increases from 2 to 8. This indicates that using more groups increases the performance of the scheduling algorithm. Across the scheduling range, speedup was recorded by the methods against the ordinary MinMin. The speedup generally improves up to a point then it begins to decline. Increasing the number of groups decreases the number of jobs per group and therefore decreases the computation time of the scheduling algorithm. Although there was a general performance improvement over the MinMin as the number of groups increases, the rate of performance improvement of a successive group over its predecessor (when using same method) decreases generally which indicates that even though there is a general performance improvement over MinMin with increasing groups, the rate of performance improvement with increasing groups does not continue to improve due to performance limiting factors like overheads with increasing group cardinality. These overheads are as a result of shared resource contention by increasing threads used by the groups in executing the scheduling algorithms.

5.4 Results, Analysis and Evaluation of the ETSB Method

This section present results and analysis of experiment 6 and experiment 7 which comprise the evaluation of the ETSB method against the ordinary MinMin (experiment 1).

5.4.1 Presentation of Results (ETSB)

The ETSB method seeks to improve on some of the drawbacks inherent in the Priority method. The method uses a sorted estimation of the processing time (or execution time) for each job to group the jobs. This method first sorts jobs based on the estimated completion times (or execution time) of jobs before applying the ETB method to distribute jobs into the groups. Sorting is done in descending order and the job with the largest completion time is placed at the top of the list and that with the least completion time placed at the bottom of the

list. The method also uses two methods for machine grouping (EvenDist and SimTog). Pairings are then made between job groups and machine groups, and then multiple instances of the MinMin scheduling algorithm is executed within paired groups (multi-scheduling) using multiple threads (multithreading) in parallel.

Table 28 and Table 29 show the result and computation of speedup of the ETSB-SimTog and ETSB-EvenDist methods over the MinMin. The MinMin used a total of 242033ms and an average of 3486.2ms to schedule the range of jobs from 1000 to 10000. With the ETSB-SimTog method, two groups took a total of 82557ms and an average of 8255.7ms to schedule same tasks. Four groups used a total of 17569ms and an average of 1756.9ms to schedule the tasks, while eight groups used a total of 3587ms and an average of 358.7ms to schedule same tasks. Likewise, with the ETSB-EvenDist method, using two groups recorded 35648ms and an average of 3564.8ms to schedule the same tasks. Four groups took a total of 4643ms and an average of 464.3ms to schedule the same range of tasks. And eight groups used 1270ms and an average of 127.0ms to schedule the same set of jobs. Figure 30 and Figure 31 show the average and total scheduling times by ETSB-SimTog and ETSB-EvenDist respectively. Figure 30 shows that for ETSB-SimTog, the MinMin took 70% of the scheduling time, 2 groups used 24% of the scheduling time, 4 groups used 5% of the scheduling time while 8 groups used just 1% of the scheduling time. Figure 31 shows that for ETSB-EvenDist, the MinMin algorithm used 85%, 2 groups used 13%, 4 groups used only 2% and 8 groups used a negligible percent time to schedule the same set of jobs.

Table 28 and Table 29 show the speedup in multiples and in percentage attained by the ETSB-SimTog and ETSB-EvenDist respectively. Both methods recorded substantial speedup in scheduling from 1000 to 10000 jobs in steps of 1000 against the MinMin. For instance, using the ETSB-SimTog method, two groups recorded a range 2.36 to 4.07 and an average of 3.21 times speedup against the MinMin. Four groups recorded a range of 10.38 to 17.07 and an average of 14.58 speedup against the MinMin and eight groups recorded a range of 34.42 to 71.04 and an average of 63.97 times speedup against the MinMin. Equally, the ETSB-EvenDist method when using two groups recorded a range of 5.95 to 8.93 and an average of 7.30 times speedup against the MinMin. Using four groups the speedup was between 27.25 to 69.46 and an average of 52.17 against the MinMin and using eight groups, the range of speedup recorded was 65.40 to 204.72 with an average of 166.69 against the MinMin. Figure 32 and Figure 33 shows the speedup in multiples (X) by the ETSB-SimTog and ETSB-

EvenDist methods while Figure 34 and Figure 35 show the speedup in percentage (%) by the ETSB-SimTog and ETSB-EvenDist methods respectively. Across all the groups, there was a general improvement in the speedup to a point after which the speedup declines.

Table 30 and Table 31show the computation of performance improvement over the MinMin and between successive groups by the ETSB-methods. The ETSB methods achieved substantial performance improvement over the MinMin as the number of groups increased from 2 to 8. For instance, the ETSB-SimTog attained 2.93 times performance improvement over the MinMin with two groups. With four groups, the method attained 13.78 times performance improvement over the MinMin. While with eight groups, the performance improved 47.48 times. In the same vein, the ETSB-EvenDist recorded 6.79 times improvement over the MinMin with two groups. With four groups, it achieved 52.12 times improvement over the MinMin while using eight groups, the performance improved to 190 times. As the scheduling changes from two groups to eight groups, the scheduling efficiency improved significantly over the MinMin. This shows that using more groups increases the performance of the scheduling algorithm. Figure 36 and Figure 37 shows the performance improvement as the number of groups increases. Figures 38 and Figure 39 show the performance improvement of the ETSB-SimTog and ETSB-EvenDist methods respectively against the MinMin and between groups. The ANOVA test (shown in Table 32) was used to check the significance of the results. All the results exhibited very low P values, showing that the differences were highly significant.

There was a general performance improvement over the MinMin with increasing groups. That was not the case when the improvements are computed between successive groups (using the same method). For instance, using the ETSB-SimTog method, the improvement of two groups over the ordinary MinMin was 2.93 times. As the group increased to 4 groups, the performance improvement of the method over the MinMin was 13.78 but the performance improvement between 2 and 4 groups was 4.70 times. As the group increased from 4 groups to 8 groups, the performance improvement over MinMin was 47.48 while an increase from 4 to 8 groups showed improvement of only 4.90 times. Between 8 groups and 2 groups, there was performance improvement of 23.02 times. Likewise with the ETSB-EvenDist method, the use of 2 groups improved performance by 6.79 times. Increasing from 2 to 4 groups, performance improved 52 times over the MinMin but improved 7.6 times between the 2 groups and 4 groups. Increasing from 4 groups to 8 groups improved performance over the

MinMin by 190 times over the MinMin, but this only brought about an improvement of 3.66 times between 4 groups and 8 groups. Also, 8 groups performed better than 2 groups by just 28 times. This decline in performance between successive groups (using the same method) is partially due to the effect of shared resource contention introduced by increasing threads used by increasing groups. For instance, two groups used two threads, four groups used four threads and eight groups used eight threads. Increase in the number of threads directly increases the effects of resource contention among threads. This impacted negatively on the performance of the method between successive groups.

Figure 40 and Figure 41 shows the declining rate of improvement between successive groups by the ETSB-SimTog and ETSB-EvenDist. This shows that even though there is a general performance improvement over MinMin with increasing groups, the rate of performance improvement with increasing groups' declines. This demonstrates that there is a limiting factor to the general performance with increasing groups within same method. Successive groups uses more threads for execution, this resulted in increased resource contention from the threads and impacted the result. Another reason for the slowing rate of improvement is the polynomial nature of theMinMin algorithm where improvement is greater when the number of jobs to be grouped is larger. As the number of groups increases, the number of jobs per group become smaller, further grouping produces smaller rates of improvement.

Table 28 Scheduling times and speedup for MinMin vs. ETSB-SimTog

Methods	MinMin vs ETSB-SimTog Scheduling time in ms				Speedup (X)			Speedup (%)		
Jobs Limit	**MinMin**	**2Grps**	**4Grps**	**8Grps**	**2Grps**	**4Grps**	**8Grps**	**2Grps**	**4Grps**	**8Grps**
1000	654	181	63	19	3.61	10.38	34.42	72.32	90.37	97.09
2000	3230	793	192	51	**4.07**	16.82	63.33	75.45	94.06	98.42
3000	7601	1876	447	110	4.05	17.00	69.10	75.32	94.12	98.55
4000	12920	3691	757	183	3.50	**17.07**	70.60	71.43	94.14	98.58
5000	18219	7706	1178	283	2.36	15.47	64.38	57.70	93.53	98.45
6000	22671	8576	1548	360	2.64	14.65	62.98	62.17	93.17	98.41
7000	29504	10343	2133	437	2.85	13.83	67.51	64.94	92.77	98.52
8000	39074	12399	2555	550	3.15	15.29	**71.04**	68.27	93.46	98.59
9000	48178	15984	3527	685	3.01	13.66	70.33	66.82	92.68	98.58
10000	59982	21008	5169	909	2.86	11.60	65.99	64.98	91.38	98.48
Total	**242033**	**82557**	**17569**	**3587**	**32.12**	**145.78**	**639.69**	**679.41**	**929.68**	**983.69**
Average	**24203.3**	**8255.7**	**1756.9**	**358.7**	**3.21**	**14.58**	**63.97**	**67.94**	**92.97**	**98.37**

Table 29 Scheduling times and speedup for MinMin vs. ETSB-EvenDist

Methods	MinMin vs ETSB-EvenDist Scheduling time in ms				Speedup (X)			Speedup (%)		
Jobs Limit	MinMin	2Grps	4Grps	8Grps	2Grps	4Grps	8Grps	2Grps	4Grps	8Grps
1000	654	110	24	10	5.95	27.25	65.40	83.18	96.33	98.47
2000	3230	372	61	30	8.68	52.95	107.67	88.48	98.11	99.07
3000	7601	851	119	51	**8.93**	63.87	149.04	88.80	98.43	99.33
4000	12920	1458	186	71	8.86	**69.46**	181.97	88.72	98.56	99.45
5000	18219	2384	333	97	7.64	54.71	187.82	86.91	98.17	99.47
6000	22671	3213	518	126	7.06	43.77	179.93	85.83	97.72	99.44
7000	29504	4605	532	152	6.41	55.46	194.11	84.39	98.20	99.48
8000	39074	6210	744	199	6.29	52.52	196.35	84.11	98.10	99.49
9000	48178	7139	949	241	6.75	50.77	199.91	85.18	98.03	99.50
10000	59982	9306	1177	293	6.45	50.96	**204.72**	84.49	98.04	99.51
Total	**242033**	**35648**	**4643**	**1270**	**73.01**	**521.72**	**1666.91**	**860.09**	**979.68**	**993.22**
Average	**24203.3**	**3564.8**	**464.3**	**127**	**7.30**	**52.17**	**166.69**	**86.01**	**97.97**	**99.32**

Table 30 Performance of ETSB-SimTog against MinMin and between groups

Methods		**ETSB-SimTog Performance Improvement(X)**			**ETSB –SimTog Performance Improvement (%)**			
	MinMin	**2Grps**	**4Grps**	**8Grps**	**Methods**	**2Grps**	**4Grps**	**8Grps**
Total	242033	82557	17569	3587		82557	17569	3587
$\frac{Total_{MinMin}}{Total_{Group}}$		**2.93**	13.78	47.48	$\frac{x_1 - x_2}{x_1} X100$ x_1 = MinMin	65.89 x_2 =2Grps	92.74 x_2 =4Grps	98.52 x_2 =8Grps
$\frac{Total_{nGrps}}{Total_{n+1Grps}}$ $n \in [2,4,8]$			**4.70**	23.02	x_1 = 2Grps		78.72 x_2 =4Grps	95.66 x_2 =8Grps
$\frac{Total_{nGrps}}{Total_{n+1Grps}}$ $n \in [2,4,8]$				**4.90**	x_1 = 4Grps			79.58 x_2 =8Grps

Table 31 Performance of ETSB-SimTog method against MinMin and between groups

Methods		**ETSB-EvenDist Performance Improvement(X)**			**ETSB –EvenDist Performance Improvement (%)**			
	MinMin	**2Grps**	**4Grps**	**8Grps**	**Methods**	**2Grps**	**4Grps**	**8Grps**
Total	242033	35648	4643	1270		35648	4643	1270
$\frac{Total_{MinMin}}{Total_{Group}}$ Better Than MinMin		**6.79**	52.12	190.57	$\frac{x_1 - x_2}{x_1} X100$ x_1 = MinMin	85.27 x_2 =2Grps	98.08 x_2 =4Grps	99.47 x_2 =8Grps
$\frac{Total_{nGrps}}{Total_{n+1Grps}}$ $n \in [2,4,8]$ Better Than 2 groups			**7.68**	28.07	x_1 = 2Grps		86.97 x_2 =4Grps	96.44 x_2 =8Grps
$\frac{Total_{nGrps}}{Total_{n+1Grps}}$ $n \in [2,4,8]$ Better than 4 groups				**3.66**	x_1 = 4Grps			72.65 x_2 =8Grps

Table 32 ANOVA results for ETSB-SimTog vs. MinMin and between group cardinality

Test	Method	P Value	Significant Difference?
1	MinMin/ ETSB-SimTog (All)	0.00423	Yes
2	MinMin/ ETSB-SimTog(2Grps)	0.0273	Yes
3	MinMin/ ETSB-SimTog(4Grps)	0.002202	Yes
4	MinMin/ ETSB-SimTog(8Grps)	0.001306	Yes
5	ETSB-SimTog(2Grps)/ ETSB-SimTog(4Grps)	0.00946	Yes
6	ETSB-SimTog(2Grps)/ ETSB-SimTog(8Grps)	0.001943	Yes
7	ETSB-SimTog(4Grps)/ ETSB-SimTog(8Grps)	0.015697	Yes

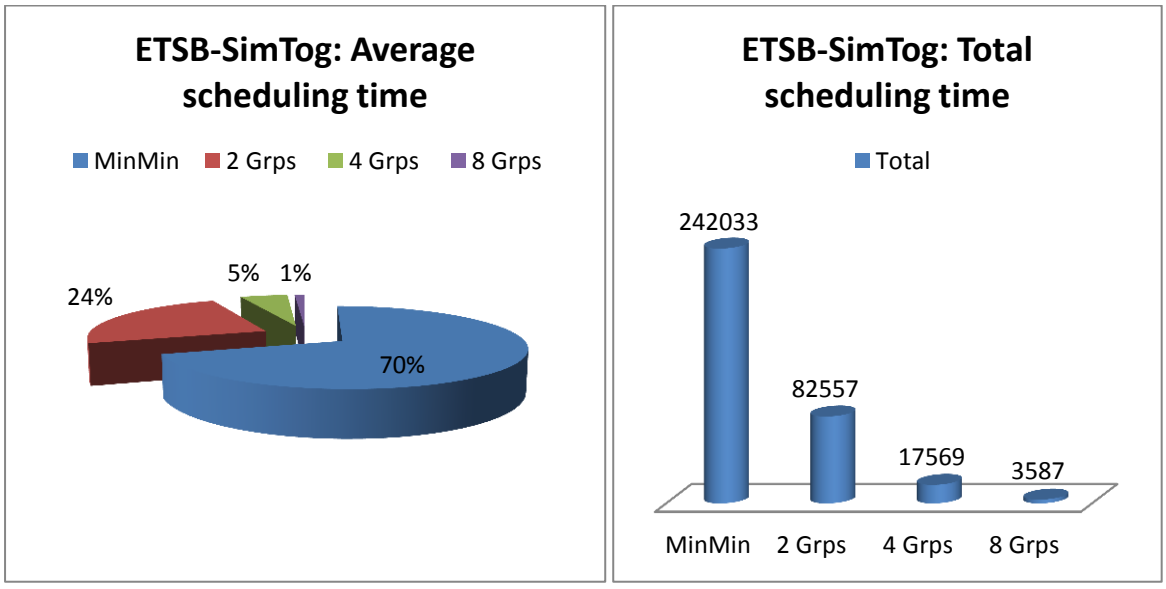


Figure 31: Total and average scheduling times of MinMin and ETSB-SimTog

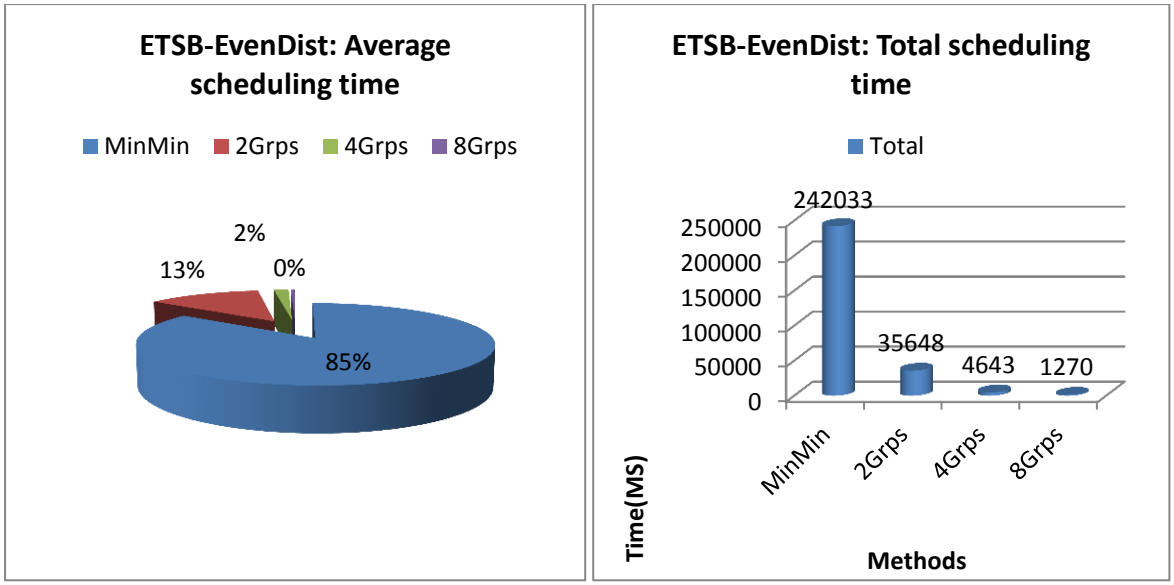


Figure 32: Total and average scheduling times of MinMin and ETSB-SimTog by groups

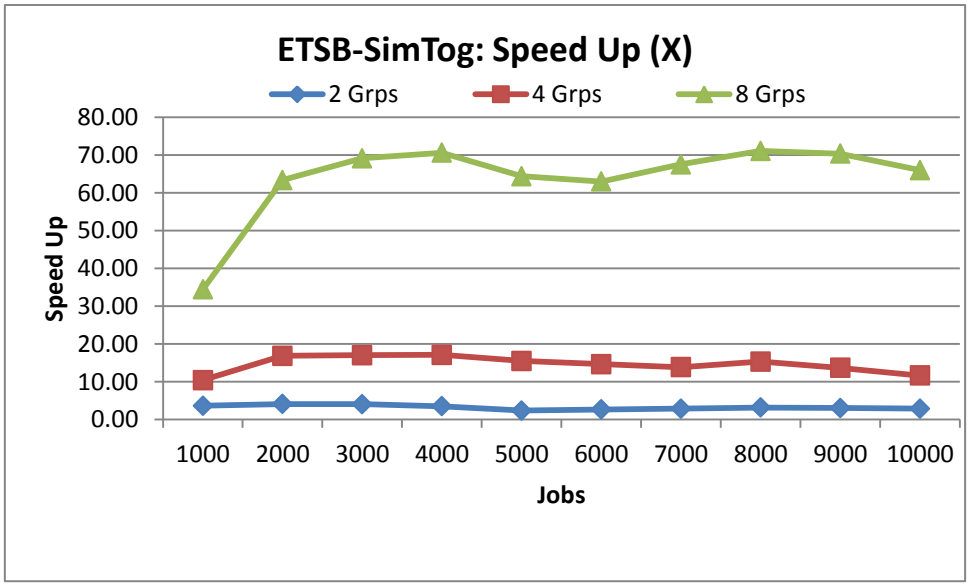


Figure 33: Speedup (in multiples) by ETSB-SimTog against MinMin

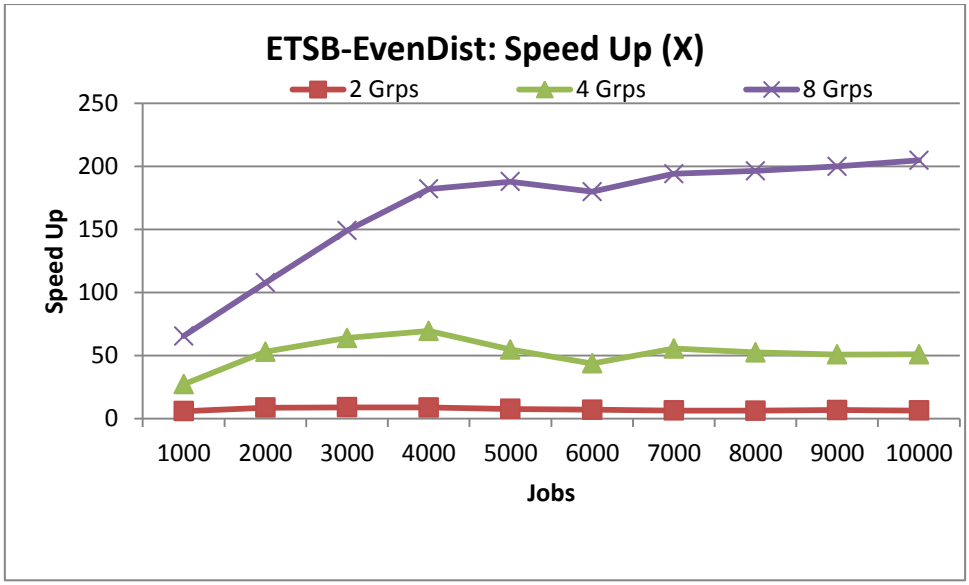


Figure 34: Speedup (in multiples) by ETSB-EvenDist over MinMin

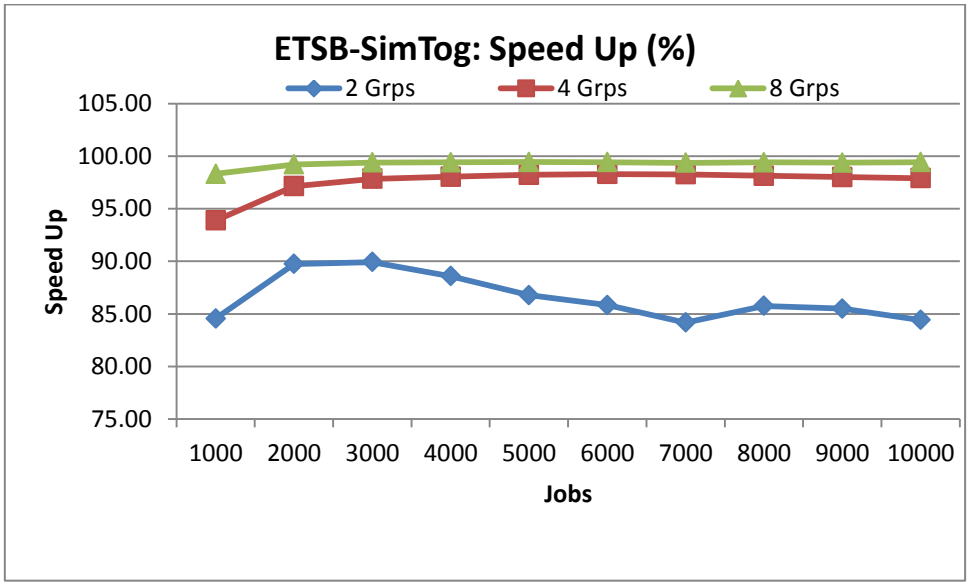


Figure 35: Speedup (in percentage) by ETSB-SimTog against MinMin

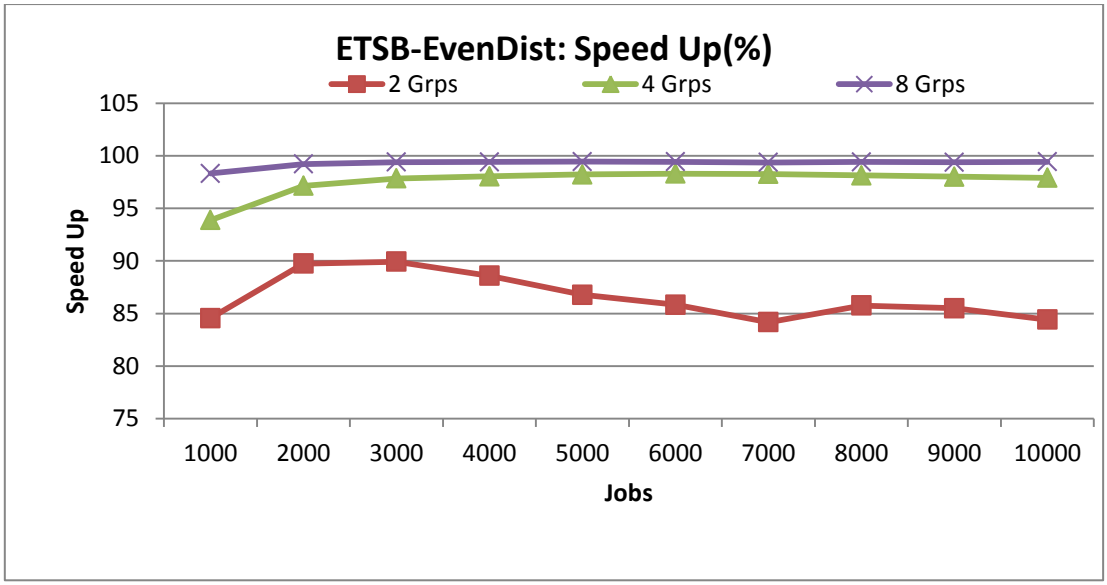


Figure 36: Speedup (in percentage) by ETSB-EvenDist against MinMin

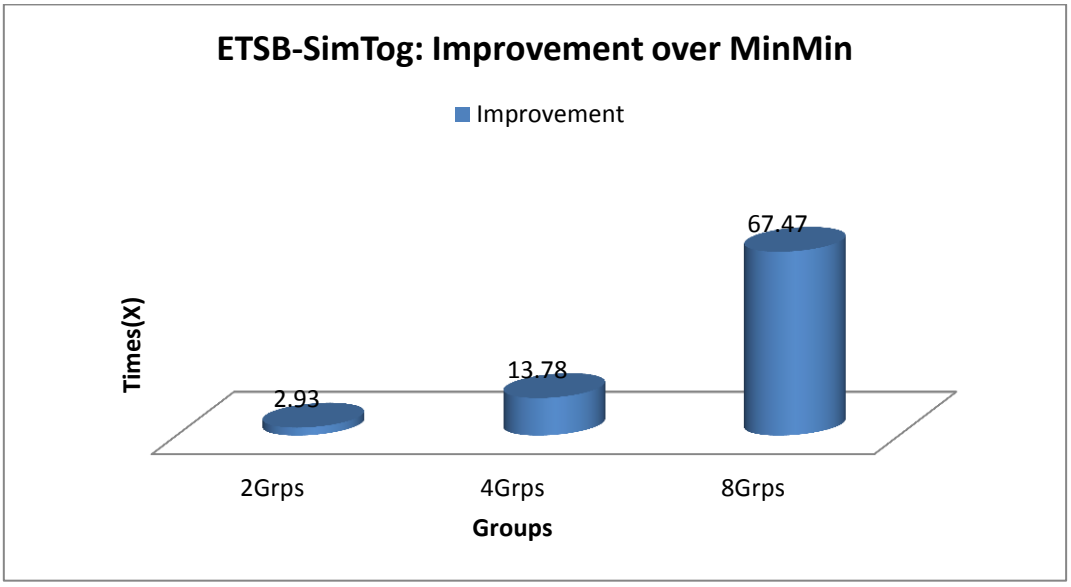


Figure 37: Improvement of ETSB-SimTog over MinMin across groups

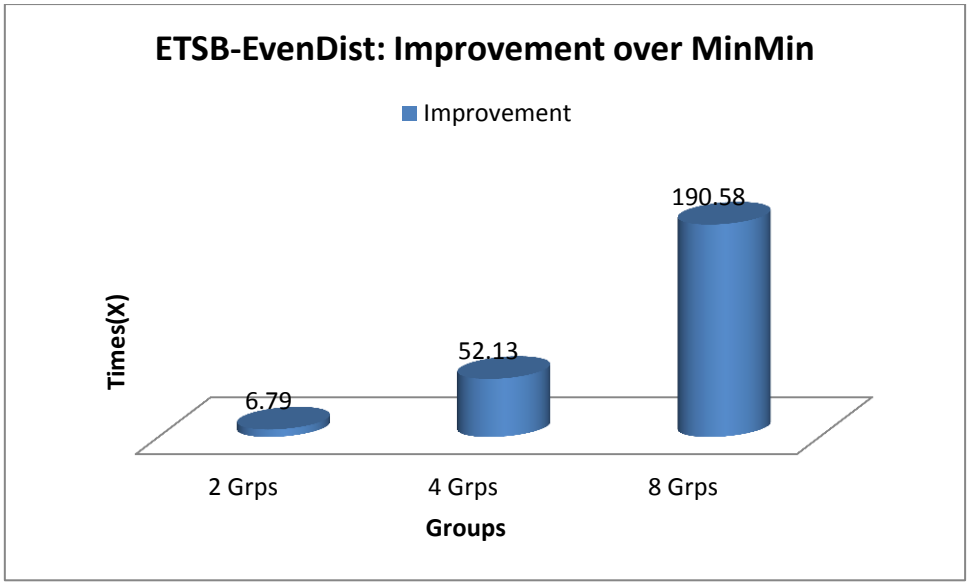


Figure 38: Improvement of ETSB-EvenDist over MinMin across groups

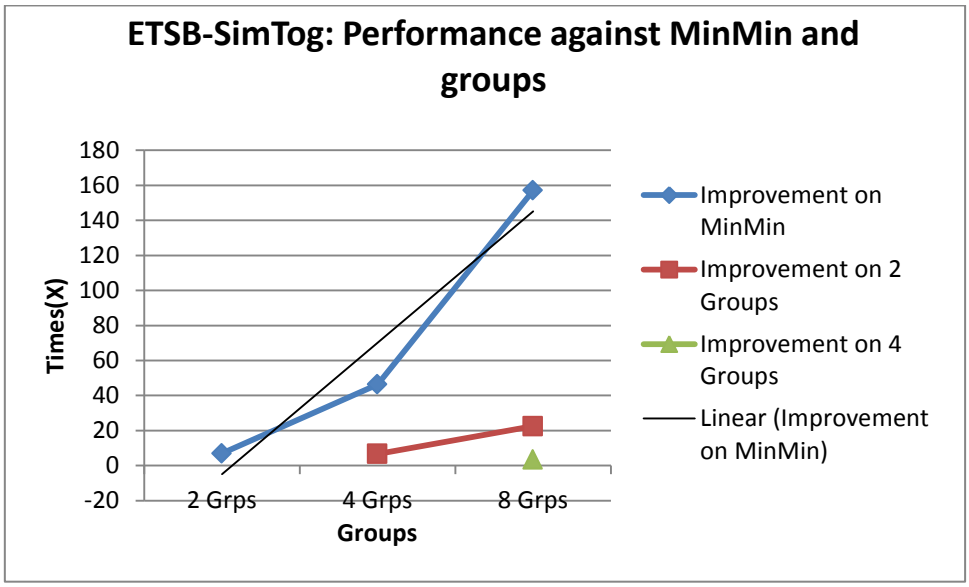


Figure 39: Improvement of ETSB-SimTog over MinMin and between groups

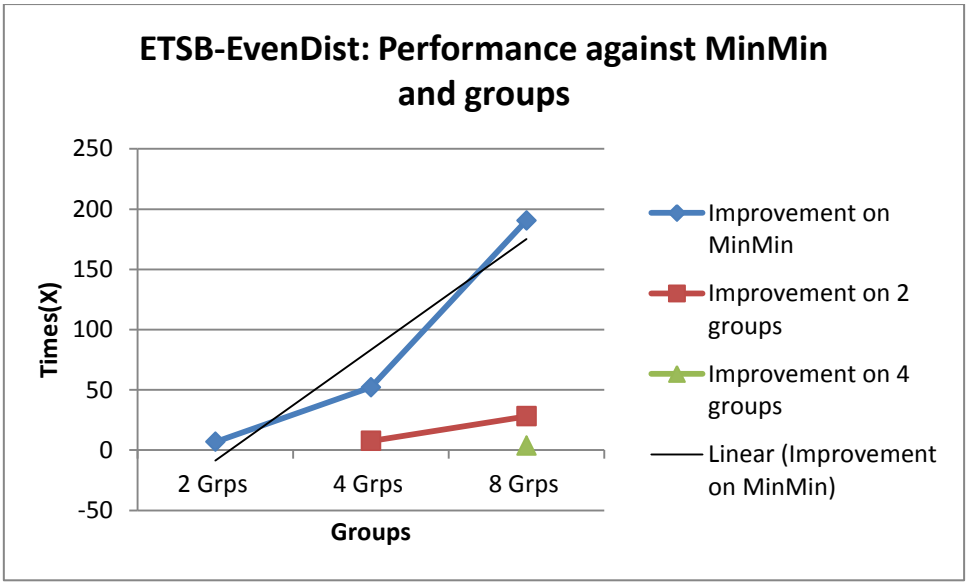


Figure 40: Performance improvement of ETSB-EvenDist over MinMin and groups

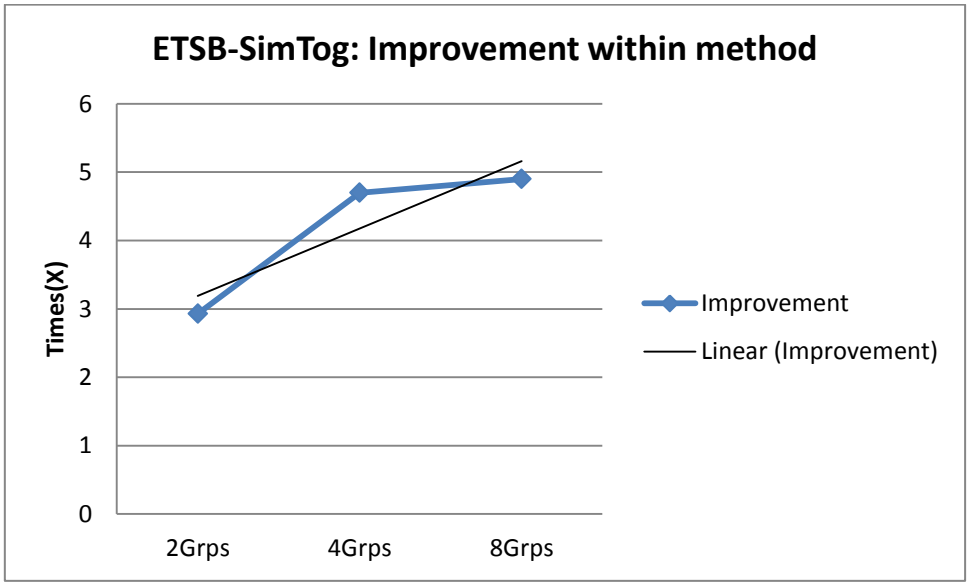


Figure 41: Rate of Improvement of ETSB-SimTog across group cardinality

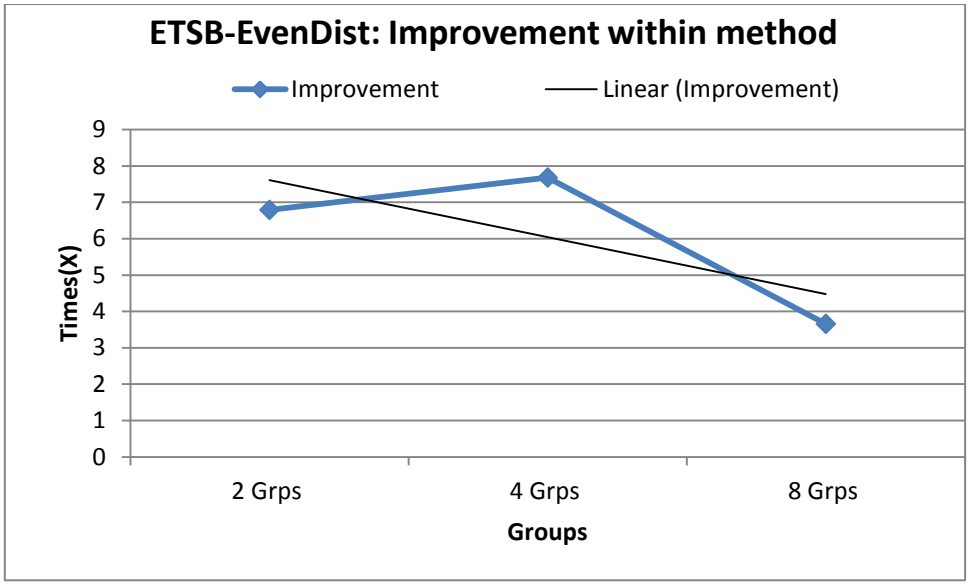


Figure 42: Rate of Improvement of ETSB-EvenDist across group cardinality

5.4.2 Discussion of Results (ETSB)

Results from the ETSB method showed significant improvement over the MinMin algorithm. The ETSB method varied the number of groups between 2, 4 and 8. Performance improved over the MinMin as the number of groups increased from 2 to 8. This indicates that using more groups increases the performance of the scheduling algorithm.

Across the scheduling range, speedup was recorded by the ETSB methods against the ordinary MinMin. The speedup generally improves up to a point then rate of improvement begins to decline. Increasing the number of groups decreases the number of jobs per group and therefore decreases the computation time of the scheduling algorithm.

However, the rate of performance improvement of each successive group over its predecessor (when using same method) decreases generally even though there was a general performance improvement over the MinMin as the number of groups increases. This indicates that even though there is a general performance improvement over MinMin with increasing groups, the rate of performance improvement with increasing groups does not continue to improve due to performance limiting factors like overheads with increasing group cardinality. These overheads are as a result partially of shared resource contention caused by increasing threads used by the groups in executing the scheduling algorithms.

5.5 Comparative Analysis of the Group-based Scheduling Methods

The previous sections discussed the results, analysis and evaluation of all the methods against the ordinary MinMin. The GPMS used three job grouping methods (Priority, ETB and ETSB) and two machines grouping methods (EvenDist and SimTog) which yielded six group scheduling methods: Priority-SimTog; Priority-EvenDist; ETB-SimTog; ETB-EvenDist; ETSB-SimTog; and ETSB-EvenDist. All grouping methods performed significantly better than non-grouping (Ordinary MinMin). Increasing the number of groups improved performance until a levelling off occurred which was apparent in all grouping methods.

This section continues the analysis of results but focuses on comparisons between the different grouping methods rather than each method against MinMin.

The Priority method used only four priority groups so comparisons of the ETB and ETSB methods to the Priority method are considered only with four groups. When comparison is between ETB and the ETSB methods, group cardinality (number of groups) is considered. In all cases, the number of threads used equals the number of groups used at that point. Hence, when number of group equals four, the number of threads also equals four.

5.5.1 Comparison between ETSB and ETB methods

This section considers the performance improvement and speedup of the ETB and ETSB methods in combination with the two machine grouping methods on a group by group basis.

5.5.5.1 Performance Improvement

Table 33 shows the scheduling times for all methods and improvements made using four groups. The ETSB-EvenDist method performed best with 52.13 times against the MinMin. The ETSB-EvenDist performed best because it guarantees even distribution of both machines and jobs. This was closely followed by the ETB-EvenDist method which recorded 51.48 times performance improvement; this was achieved due to the even distribution of jobs guaranteed by the EvenDist method. The ETB-SimTog was next with 46.33 times performance improvement against the MinMin, and the ETSB-SimTog was the least with

13.78 improvements. Table 40 shows the aggregate mean improvement and average improvement by all methods and by all groups. Two Groups made an average of 6.16 improvements across all methods which represent 4% of the general improvement. Four groups made an average of 40.13 improvements across all methods representing 22% of the general improvement and 8 groups made an average of 141 improvements across all the methods which represent 74% of the general improvement. On the other hand, across the groups, the ETSB-EvenDist performed best with an average of 83.16 improvements across all groups. This represents 33% of overall improvement. This was followed by the ETB-EvenDist with an average improvement of 70.14 times, representing 28% of overall improvement. Next is the ETB-SimTog with improvement of 70.12 times representing 28% of general improvement. The ETSB-SimTog came worst with an average improvement of 27.25 times representing 11% of general improvement. This is due to the effect of distributing jobs equally (by the ETSB method) to unbalanced (SimTog) machine groups. Figure 50 shows the percentage performance by the methods across groups and Figure 51 shows the percentage performance by the groups across methods.

Figure 46 shows the improvement of the methods using 2 groups. Figure 48 shows the improvement by the methods using 4 groups. Figure 50 shows the improvements by the methods when using 8 groups. Figure 51 shows the percentage performance by the methods across groups and Figure 52 shows the percentage performance by the groups across methods. In these figures, ETSB-EvenDist stands out as showing the highest performance improvement and ETSB-SimTog as showing the worst. Based on the group by group analysis, using 2 groups, the ETB-EvenDist method performed better than the rest, followed by the ETSB-EvenDist and ETB-SimTog. The ETSB-SimTog performed worse than the other methods. Using 4 groups and 8 groups; the ETSB-EvenDist method performed better followed by ETB-EvenDist and ETB-SimTog methods. The ETSB-SimTog performed worse in all the groups. Generally, there is a remarkable increase in performance with increase in the number of groups. The ETSB-EvenDist method performed best because it ensures load-balancing by evenly distributing both jobs and machines among the groups. While the ETSB-SimTog performed worse because the machines in the groups were unbalanced and the total scheduling time of the worst machine group impacted the overall scheduling time.

5.5.5.2 Speedup

Table 37 and Figure 44 show result and graph of speedup of ETB and ETSB methods using 2 groups. The ETB-SimTog method showed a better speedup than the other methods at some points than the other methods, while the ETSB-SimTog showed the worst speedup compared to the rest. Table 38 and Figure 46 show result and graph of speedup by ETB and ETSB methods using 4 groups. The ETB-SimTog showed a better speedup than the other methods to a point (when jobs = 5000). Thereafter, the ETB-EvenDist method picked up and showed higher speedup. The ETSB-SimTog method showed less speedup than the other three methods. Table 39 and Figure 48 show results and graph of speedup by ETB and ETSB methods using 8 groups. The ETB-SimTog, ETSB-EvenDist and ETB-EvenDist performed relatively equally to a point (when jobs = 5000). Beyond this point, the ETSB-EvenDist method showed the best speedup closely followed by the ETB-EvenDist.

Based on the group by group analysis, using 2 groups, the ETB-SimTog had better speedup than the other methods; this was followed by the ETB-EvenDist and the ETSB-EvenDist. The ETSB-SimTog performed worst compared to the other methods. The result of using 2 groups contrast with those of 4 groups and 8 groups. Splitting the machines into just two groups based on configuration favoured the SimTog method more. Using 4 groups and 8 groups; the ETSB-EvenDist had the best speedup, this was followed by the ETB-SimTog and the ETB-EvenDist. The ETSB-SimTog had the worst speedup. From this 4 and 8 groups analysis, it can be deduced that the ETSB method which guarantees fairer even distribution of jobs among the groups when paired with the EvenDist method that also guaranteed equitable distribution of machines among the groups enhances speedup more than when paired with the SimTog method that does not support fair distribution of jobs.

5.5.2 Comparison between Priority, ETB and ETSB methods

This section compares the results of the ETB and ETSB methods against results of the Priority method.

Table 33 shows scheduling results for MinMin and the other methods using four groups. Using the Priority method, the SimTog and EvenDist methods recorded scheduling times of 41006ms and 35807ms which represents a performance improvement of 5.90 and 6.76 respectively over the MinMin. While using the ETSB method, the SimTog and EvenDist methods recorded scheduling times of 17569ms and 4643 ms representing a 13 times and 52 times performance improvement over the MinMin respectively. With the ETB method, the SimTog and EvenDist methods recorded 5224ms and 4701 ms, yielding 46 times and 51 times performance improvement respectively over the MinMin. It is clear that ETB and ETSB methods performed better than the Priority method.

With Priority, both machine grouping methods (EvenDist and SimTog) were observed to have recorded the highest speedup against MinMin at the point when the number of jobs equals 4000. Using the ETSB-EvenDist method, 2, 4 and 8 groups recorded its highest speedup at points 3000, 4000, and 10000 respectively with values of 8.9, 69 and 204 respectively. Using the ETB-SimTog, 2, 4 and 8 group recorded highest speedup at points 4000, 3000 and 5000 respectively with speedup values of 11, 76 and 187 respectively.

In Figure 53 and Figure 54, comparison was made of the improvement by all the GPMS methods using four groups. It shows that the ETSB-EvenDist, ETB-EvenDist, ETB-SimTog, ETSB-SimTog, Priority-EvenDist and Priority-SimTog methods achieved 52, 51, 46, 13, 6 and 9 times improvements respectively over the MinMin. These values represent a total percentage improvement of 30%, 29%, 26%, 8%, 4%, and 3% respectively.

Figure 55 shows the mean scheduling time and percentage mean scheduling time of the GPMS methods. It shows that the Priority, ETSB and ETB had a mean of 3840.7ms, 1110.9ms and 496.4 ms respectively. These values further represent a total percentage of 71%, 20% and 9% respectively by the methods. These results indicate that ETSB and ETB perform better than Priority. The ETSB-EvenDist and ETB-EvenDist method performed better than other methods because the method guaranteed that both jobs and machines were

equally shared among the groups. The Priority-SimTog method performed worse because both jobs and machines were not equally balanced into the groups.

The ANOVA test results generally back up the observation that ETSB and ETB perform better than Priority (see Table 35) showing significant differences between ETSB and ETB on the one hand and Priority on the other. The only exception is Priority vs. ETSB, where the significance is marginal, right on the P=0.05 boundary (see Table 35, Test 3). A closer inspection reveals that ETSB-SimTog has the least improvement among the GPMS methods. There was no significant difference between Priority-SimTog and ETSB-SimTog (see Table 35, Test 7). In Figure 53, the performance of the ETSB-SimTog method is much closer to that of the Priority methods than any other GPMS method. Overall though, the ETB and ETSB methods perform better than the Priority method. The ANOVA analysis of the Priority vs. ETB and ETSB methods combined gave a P value of 0.027992 which shows that the difference is significant (see Table 35, Test 1).

The ETB and ETSB methods performed much better than the Priority method because with the Priority method, the jobs were not evenly distributed into groups. This resulted in most jobs getting sorted into one group. When job groups are assigned to machine groups such an uneven distribution can result in a particular machine group being overloaded. Scheduling from that group therefore tends towards the same execution time of the ordinary MinMin method. Hence, the general performance of the Priority method was affected. Furthermore, the MinMin scheduling time tends to polynomial (Freund et al. 1998) which means that increase in the number of instances of the input set increases the time per instance directly. Hence, smaller groups have smaller time per instance and by extension smaller scheduling time and larger groups have a comparatively inflated scheduling time which impacted the total scheduling time of the method. Although in some cases Priority might work equally well as ETB or ETSB, this cannot be guaranteed unless the priority allocations scheme guarantees equity in job distribution.

5.5 Statistical Tests

This section discusses the statistical analysis carried out on the results from the experiments.

Analysis of variance

The ANOVA significance test results for the various performances are shown in Table 35. Significant differences between results from the GPMS method and the MinMin are shown. Furthermore, significant difference was shown between the Priority method and the ETB and ETSB methods. Figure 42 illustrates the difference in performance between MinMin and the ETB and ETSB grouping methods. The grouping methods perform much better than ordinary MinMin. The ANOVA results show these differences to be significant. Figure 43 illustrates the difference in performance between MinMin and the ETB and ETSB grouping methods more clearly. Figures 44, 46 and 48 compare these grouping methods without MinMin and with differing numbers of groups. The ETSB-SimTog method performs worse than the others. The ANOVA results, which are discussed in the next paragraph, show this performance difference to be significant. There was no significant difference between the performances of the ETB and ETSB grouping methods.

All GPMS grouping methods performed better than MinMin with significant differences shown in the ANOVA results. Test 1 in Table 36 used the mean scheduling speed of all three GPMS methods and compared this to MinMin and a significant difference is shown. This shows that overall the GPMS performs significantly better than the ordinary MinMin. The significance analysis shows that there was no significant difference between ETB and ETSB grouping methods (Table 36, Test 8), both of which performed significantly better than MinMin. However a difference is shown between ETB-SimTog vs. ETSB-SimTog; this indicates that (using the same machine grouping method) the job grouping methods (ETB and ETSB) employed have different effects. Also, a significant difference was shown between ETSB-EvenDist vs. ETSB-SimTog. This indicates that the machine grouping methods (EvenDist and SimTog) also have different effects on the result when the job grouping method is the same. For instance, the SimTog method was not as effective as the EvenDist.

Standard Deviation

Table 41 shows the analysis of standard deviation, correlation and t-test. The standard deviation analysis was carried out to determine how widespread the data are from the mean.

Standard deviation of the methods

The standard deviation for the MinMin algorithm =**19831.78** with mean of **24203.3.** The standard deviation for the PrioritySimTog method = **4085.54** and very close to the mean of **4100.6.** The standard deviation for the PriorityEvenDist method =**3845.52**, is greater and close to the mean of **3580.7**. The standard deviation for the ETBEvenDist method = **396.71** close to the mean of **470.1**. The standard deviation for the ETSBEvenDist method = **394.27** close to the mean of **464.30**. The standard deviation for the ETBSimTog method = **466.46** close to the mean of **522.4** and the standard deviation for the ETBSimTog method = **466.46** close to the mean of **522.4**. The closeness of the standard deviation to the mean by all the GPMS methods shows that the results across the methods are reliable and consistent.

Correlation

The correlation analysis was carried out to determine the strength of relationships or randomness between the results from the different methods. A correlation of 1 indicates that the results are strongly related. Values close to 1 also indicate strong relationship while values further away from 1 indicates less relationship or randomness between the results.

From the computation, all the results from the various methods are strongly correlated with values of 0.9xx. For instance, the correlation between the MinMin and the ETB-EvenDist method is 0.9935. The correlation between the MinMin and the ETSB-EvenDist method is 0.9953. The correlation between the MinMin and the ETB-SimTog method is 0.9928 and between the MinMin and the ETSB-SimTog, the correlation is 0.9885. The correlation between the Priority method and the ETB-SimTog and ETB-EvenDist methods are 0.9903 and 0.9908 respectively. The correlation between the Priority and the ETSB-SimTog and the ETSB-EvenDist are 0.9891and 0.9744 respectively. Furthermore, the correlation between the ETB-SimTog and ETSB-SimTog method is 0.9872 and the correlation between the ETB-EvenDist and the ETSB-EvenDist is 0.9886. These values of 0.9xxx are very close to 1 and

indicate a very strong correlation (relationship) between the results. See Table 41 column 3 and column 6. This strong correlation between all the results by the GPMS methods indicates that the results are reliable and not random. It also strengthens the argument that grouping jobs before scheduling in parallel can increase the scheduling efficiency of scheduling algorithms and means that the same outcomes are achievable if the method is generalised and applied in real systems.

T-Test

The t-test was carried out to also reveal if there are significant differences between results from the methods. The t- tests shows a very significant value of 0.003xxx between the MinMin and the ETB and ETSB methods. The result between the Priority method and the ETB and ETSB methods is also significant with a value of 0.019127 for ETB-EvenDist, 0.0192 for ETSB-EvenDist, 0.0123 for ETB-SimTog and 0.0152 for ETSB-SimTog. The t-test value of 0.7658 between the ETB-EvenDist and the ETSB-EvenDist is not significant while the t-test value of 0.01533 between ETB-SimTog and ETSB-SimTog is significant and confirms the ANOVA test. See Table 41 column 4 and column 7.

Table 33 Results and performance by GPMS methods

Method	MinMin	Priority		ETB		ETSB	
Jobs	MinMin	EvenDist	SimTog	EvenDist	SimTog	EvenDist	SimTog
1000	654	95	105	40	32	24	63
2000	3230	340	412	92	50	61	192
3000	7601	673	839	163	99	119	447
4000	12920	1092	1345	252	196	186	757
5000	18219	1776	2008	323	324	333	1178
6000	22671	2837	3339	383	522	518	1548
7000	29504	3860	4570	511	703	532	2133
8000	39074	5312	7500	729	907	744	2555
9000	48178	7818	8830	954	992	949	3527
10000	59982	12004	12058	1254	1399	1177	5169
Total	242033	35807	41006	4701	5224	4643	17569
Ave	24203.3	3580.7	4100.6	470.1	522.4	464.3	1756.9
StanDev	19831.78	3845.53	4085.54	396.71	466.46	394.27	1631.86
Performance Improvement(X)		**6.76**	**5.90238**	**51.48**	**46.33**	**52.13**	**13.78**

Table 34 Result and Improvement for ETB and ETSB

Method		ETB		ETSB	
Jobs	MinMin	EvenDist	SimTog	EvenDist	SimTog
1000	654	40	32	24	63
2000	3230	92	50	61	192
3000	7601	163	99	119	447
4000	12920	252	196	186	757
5000	18219	323	324	333	1178
6000	22671	383	522	518	1548
7000	29504	511	703	532	2133
8000	39074	729	907	744	2555
9000	48178	954	992	949	3527
10000	59982	1254	1399	1177	5169
Total	**242033**	**4701**	**5224**	**4643**	**17569**
Ave	**24203.3**	**470.1**	**522.4**	**464.3**	**1756.9**
Performance Improvement		**51.48**	**46.33**	**52.13**	**13.78**
Cumulated average		**48.91**		**32.96**	

Table 35 ANOVA Test: Priority vs. ETB and ETSB methods

Test	Method	P value	Significant Difference?
1	Priority vs. GPMS (ETB and ETSB averaged)	0.027992	Yes
2	Priority vs. ETB	0.015965	Yes
3	Priority vs. ETSB	0.048583	Marginal – Yes/No?
4	Priority-EvenDist vs. ETB-EvenDist	0.020335	Yes
5	Priority-SimTog vs. ETB-SimTog	0.013124	Yes
6	Priority EvenDist vs ETSB EvenDist	0.020128	Yes
7	Priority SimTog vs ETSB-SimTog	0.109315	No

Table 36 ANOVA Test: MinMin, ETB and ETSB methods

Test No	Method	P value	Significant Difference? (Threshold level: P = 0.05)
1	MinMin vs. GPMS	0.001537	Yes
2	MinMin vs. ETB	0.001373	Yes
3	MinMin vs. ETSB	0.001723	Yes
4	MinMin vs. ETB-EvenDist	0.00136	Yes
5	MinMin vs. ETB-SimTog	0.001387	Yes
6	MinMin vs. ETSB-EvenDist	0.001357	Yes
7	MinMin vs. ETSB-SimTog	0.010622	Yes
8	ETB vs. ETSB	0.093828	No
9	ETB-EvenDist vs. ETSB-EvenDist	0.974201	No
10	ETB-SimTog vs. ETSB-SimTog	0.033619	Yes
11	ETB- EvenDist vs. ETB-SimTog	0.790165	No
12	ETSB-EvenDist vs. ETSB-SimTog	0.025532	Yes
13	SimTog vs. EvenDist	0.073511	No

Table 37 Speedup for ETB and ETSB methods using two groups

JobsLimit	ETB-SimTog	ETSB-EvenDist	ETSB-SimTog	ETB-EvenDist
1000	6.41	5.95	3.61	6.47
2000	8.71	8.68	4.07	9.76
3000	10.20	8.93	4.05	9.92
4000	11.10	8.86	3.50	8.76
5000	9.80	7.64	2.36	7.56
6000	8.47	7.06	2.64	7.06
7000	7.29	6.41	2.85	6.32
8000	7.54	6.29	3.15	7.02
9000	6.63	6.75	3.01	6.89
10000	5.33	6.45	2.85	6.42
Sum	**81.47**	**73.02**	**32.09**	**76.18**

Table 38 Speedup for ETB and ETSB methods using four groups

JobsLimit	ETB-SimTog	ETSB-EvenDist	ETSB-SimTog	ETB-EvenDist
1000	20.44	27.25	10.38	16.35
2000	64.60	52.95	16.82	35.11
3000	**76.78**	63.87	17.00	46.63
4000	65.92	**69.46**	**17.07**	51.27
5000	56.23	54.71	15.47	56.41
6000	43.43	43.77	14.64	**59.19**
7000	41.97	55.46	13.83	57.74
8000	43.08	52.52	15.29	53.60
9000	48.57	50.77	13.66	50.50
10000	42.87	50.96	11.60	47.83
Sum	**503.89**	**521.72**	**145.76**	**474.63**

Table 39 Speedup for ETB and ETSB methods using eight groups

JobsLimit	ETB-SimTog	ETSB-EvenDist	ETSB-SimTog	ETB-EvenDist
1000	65.40	65.40	34.42	59.45
2000	115.36	107.67	63.33	129.20
3000	165.24	149.04	69.10	165.24
4000	184.57	181.97	70.60	170.00
5000	**187.82**	187.82	64.38	**182.19**
6000	131.05	179.93	62.98	177.12
7000	133.50	194.11	67.51	159.48
8000	138.56	196.35	**71.04**	171.38
9000	167.28	199.91	70.33	163.87
10000	183.99	**204.72**	65.99	175.39
Sum	**1472.77**	**1666.92**	**639.68**	**1553.32**

Table 40 Groups aggregate mean improvement

No.	Grouping Method	Groups average speedup (or mean improvement)		
		2 Grps	**4 Grps**	**8 Grps**
1	ETSB-SimTog	3.21	14.58	63.97
2	ETSB-EvenDist	6.79	52.13	190.57
3	ETB-SimTog	6.98	46.33	157.06
4	ETB-EvenDist	7.62	47.46	155.33
Aggregate mean improvement (over MinMin)		6.15	40.125	141.7325
Aggregate mean improvement (between groups)			6.52439	3.532274

Table 41 Standard Deviation, Correlation and t-tests for Priority, ETB and ETSB

No.	Evenly Distributed Methods			Similar Together Methods		
	Standard Deviation For	Correlation (between)	t-test (between)	Standard Deviation for	Correlation (between)	t-test (between)
1	ETB-EvenDist = 396.71 and very close to the mean of 470.1	MinMin and ETB = 0.9935	MinMin and ETB = 0.003841	ETB-SimTog = 466.46 (and close to the mean of 522.4)	MinMin and ETB = 0.9928	MinMin and ETB = 0.00381
2	ETSB-EvenDist = 394.27 and very close to the mean of 464.30	MinMin and ETSB = 0.9953	MinMin and ETSB = 0.003837	ETSB-SimTog = 1631.86 and close to the mean of 1756.9	MinMin and ETSB = 0.9885	MinMin and ETSB = 0.003643
3		Priority and ETB = 0.990792	Priority and ETB = 0.019127		Priority and ETB = 0.990352	Priority and ETB = 0.012275
4	Priority-EvenDist = **3845.52. G**reater and close to the mean of **3580.7**	Priority and ETSB = 0.974364	Priority and ETSB = 0.019208	Priority-SimTog =**4085.54.** Less and very close to the mean of 4100.6	Priority and ETSB = 0.989133	Priority and ETSB = 0.015329
5		ETB and ETSB = 0.988611	ETB and ETSB = 0.765807		ETB and ETSB = 0.98718	ETB and ETSB = 0.015329

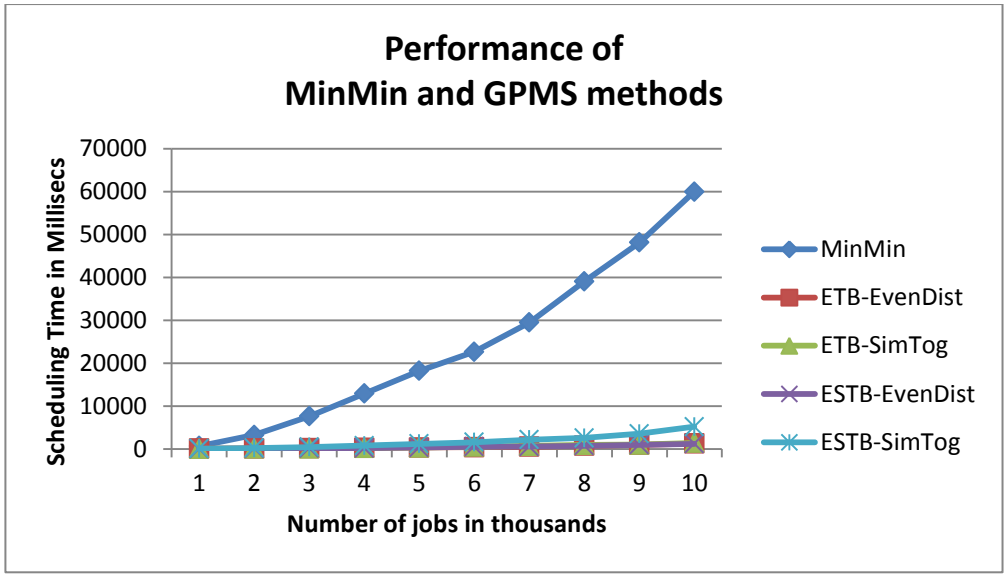


Figure 43: Scheduling performance by all methods with increasing jobs

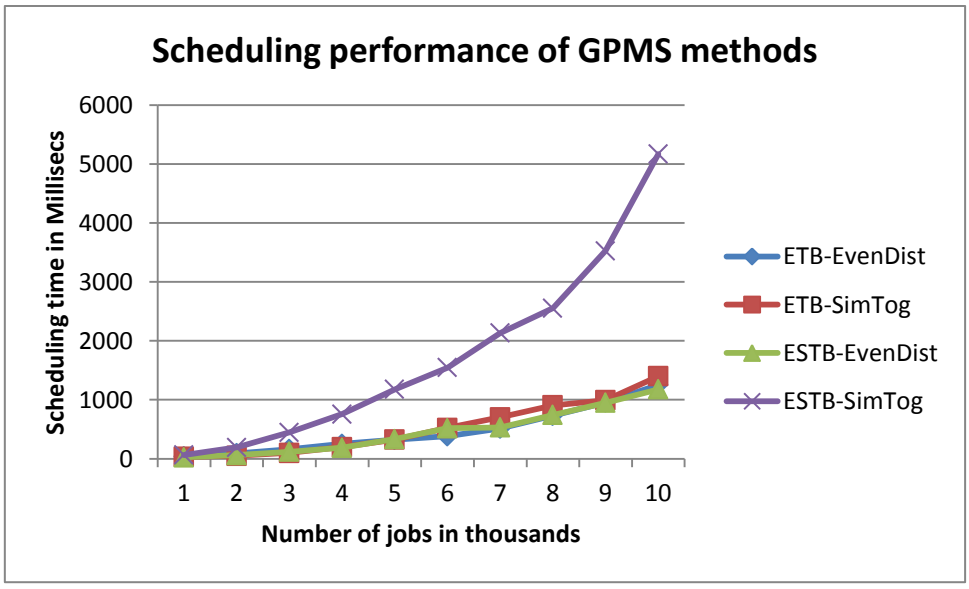


Figure 44: Scheduling performance by GPMS methods with increasing jobs

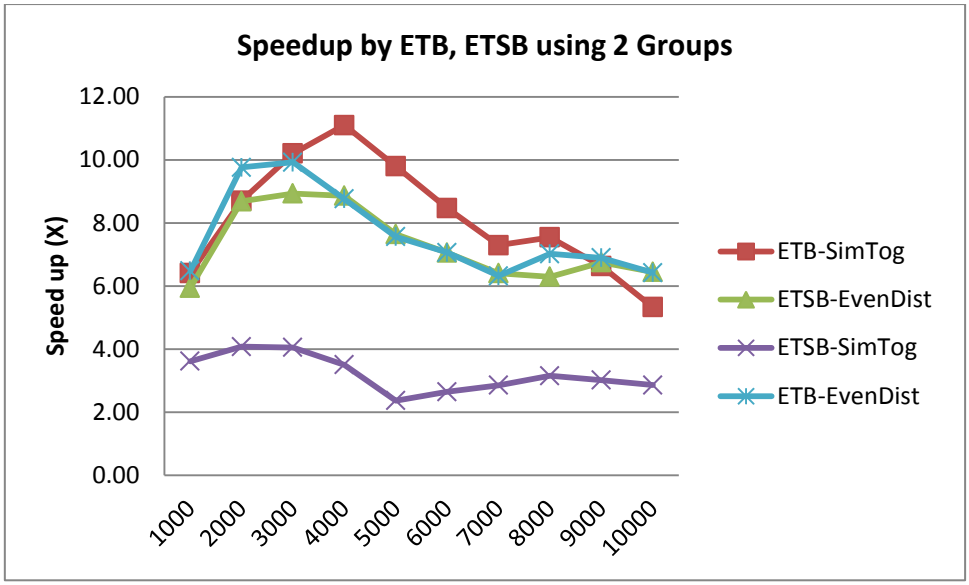


Figure 45: Speedup by ETB and ETSB methods using two groups

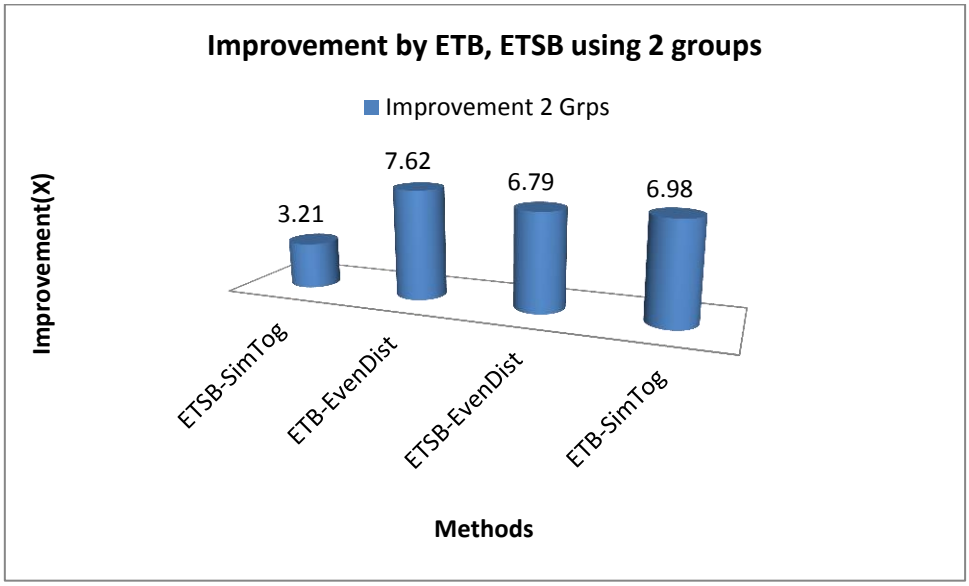


Figure 46: Improvement across methods using two groups

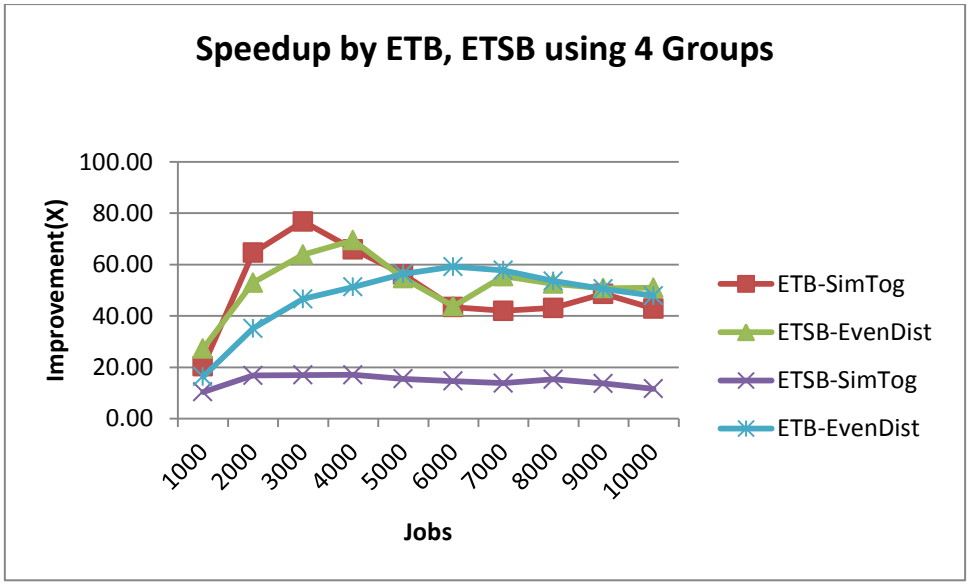


Figure 47: Speedup by ETB and ETSB methods using four groups

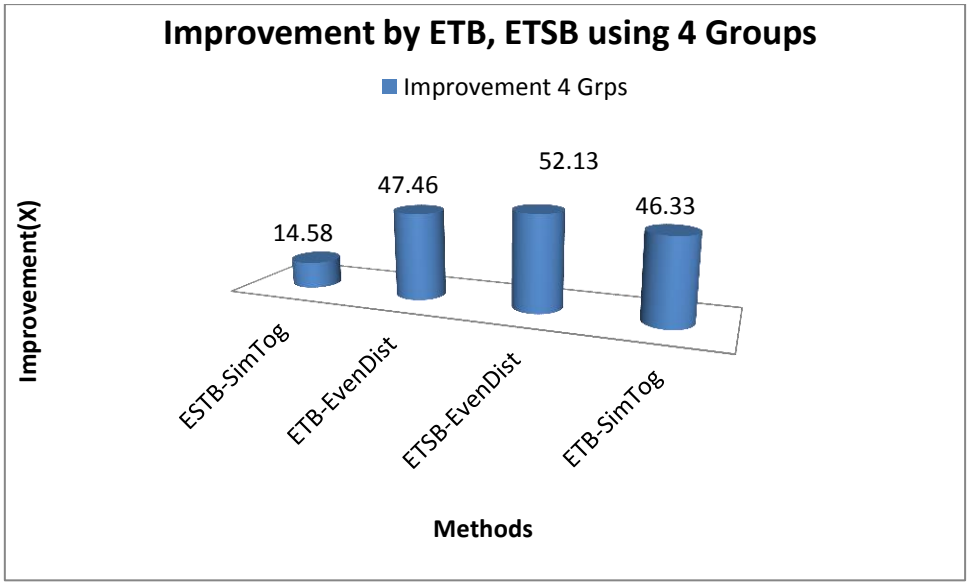


Figure 48: Improvement across methods using four groups

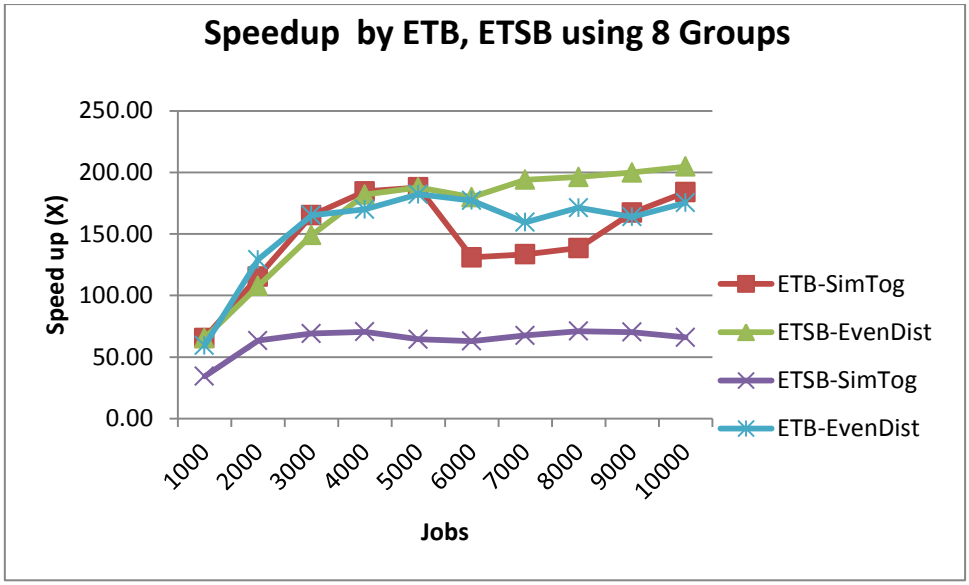


Figure 49: Speedup by ETB and ETSB methods using eight groups

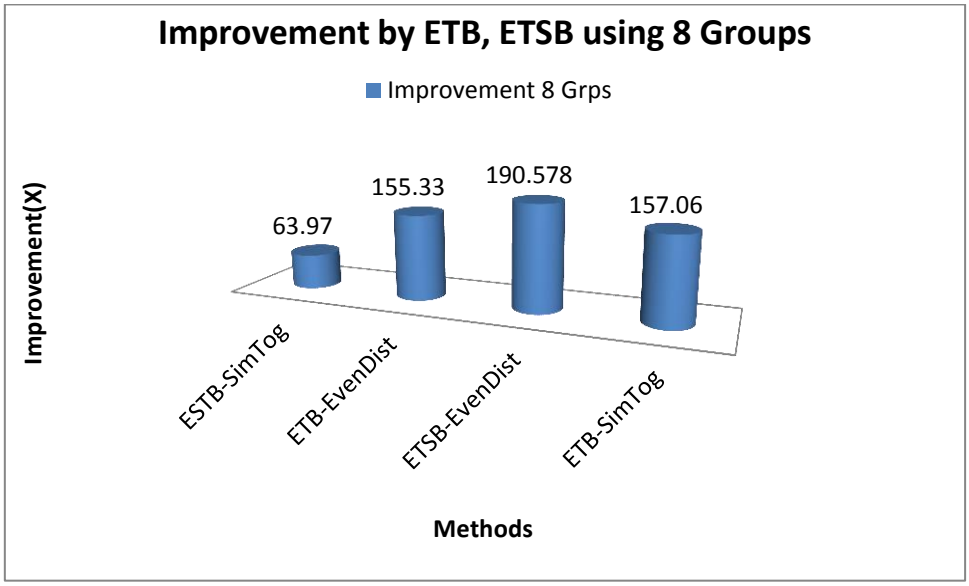


Figure 50: Improvement across methods using eight groups

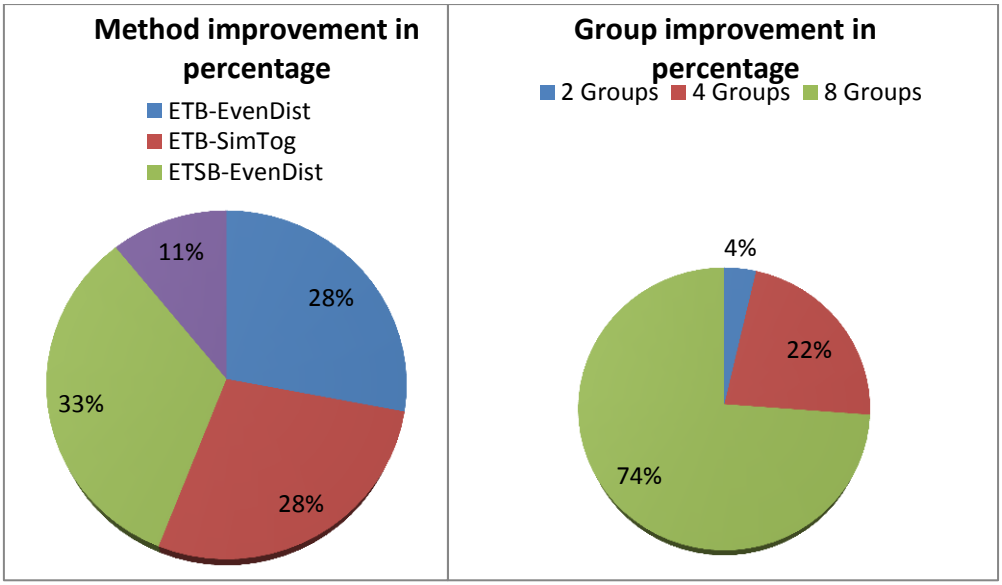


Figure 51: Percentage improvement by ETB and ETSB methods and by groups

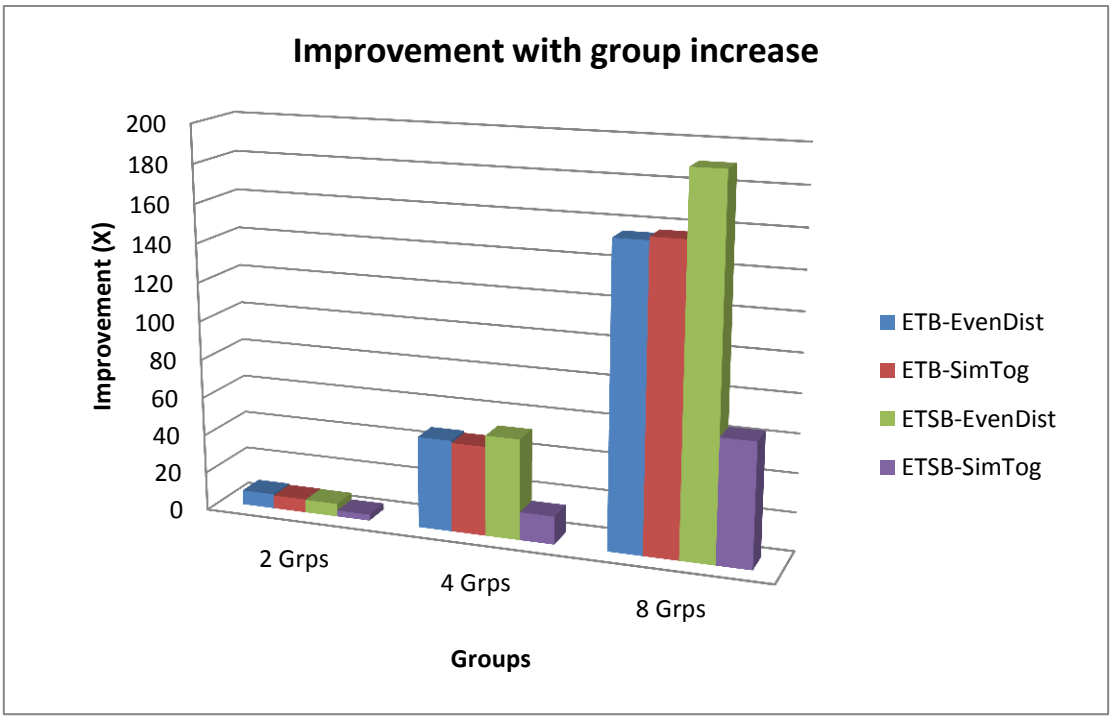


Figure 52: Improvement by ETB and ETSB methods across Groups

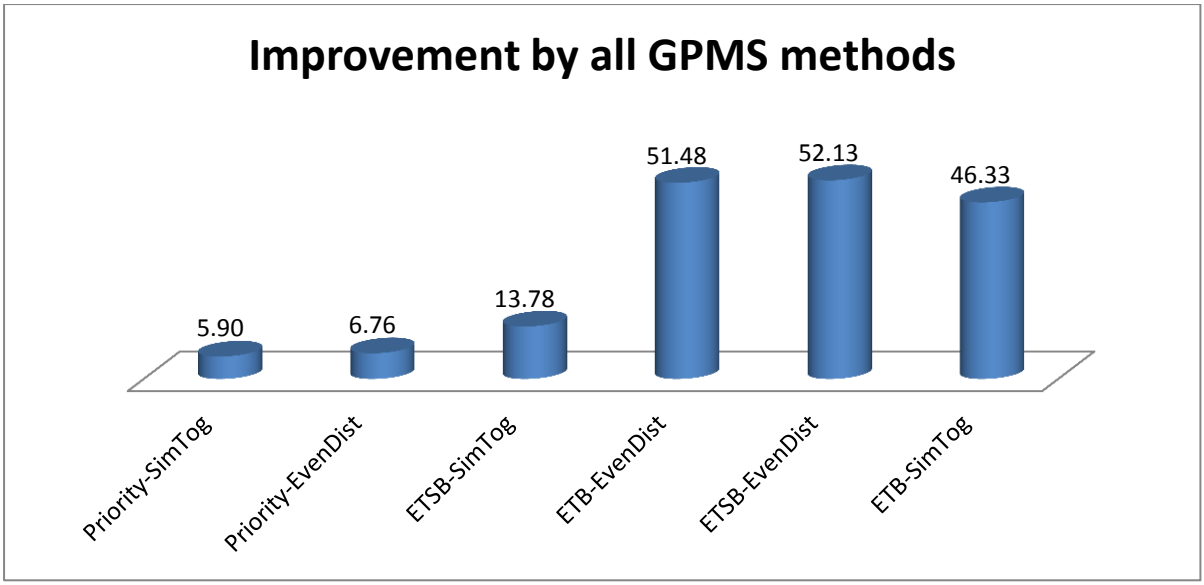


Figure 53: Improvement comparison between the GPMS methods (multiples)

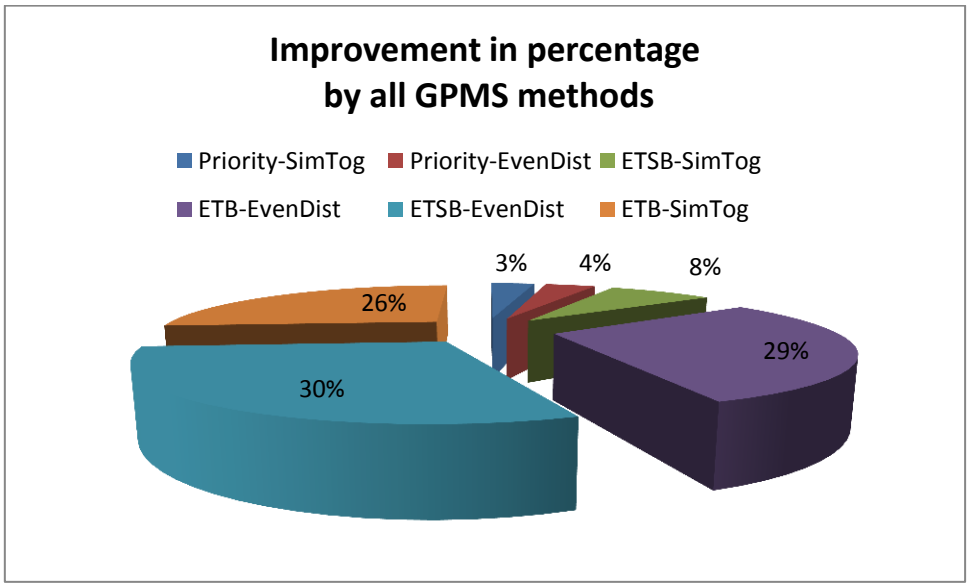


Figure 54: Improvement comparison between the GPMS methods (percentage)

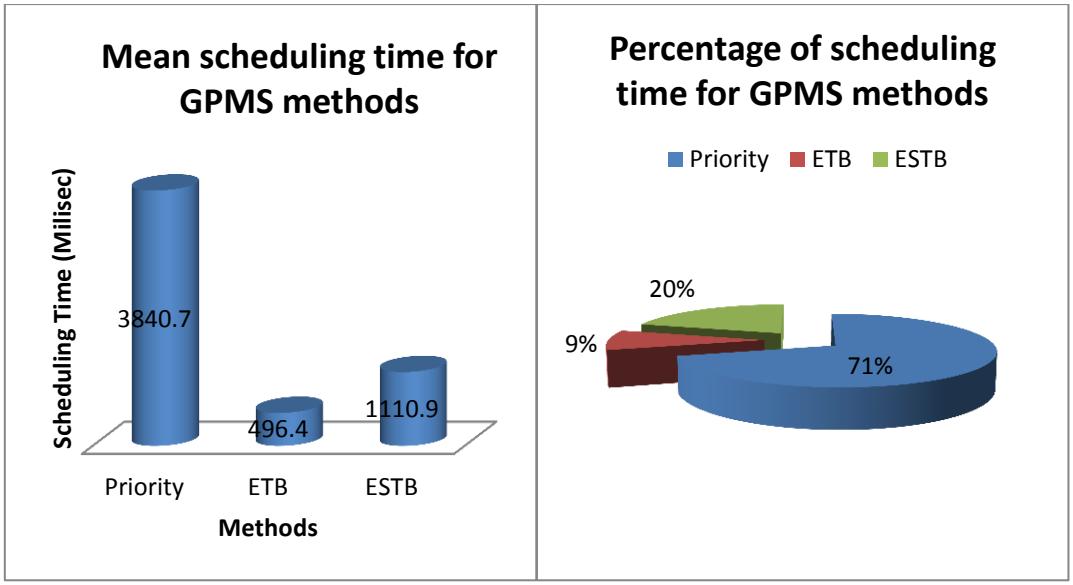
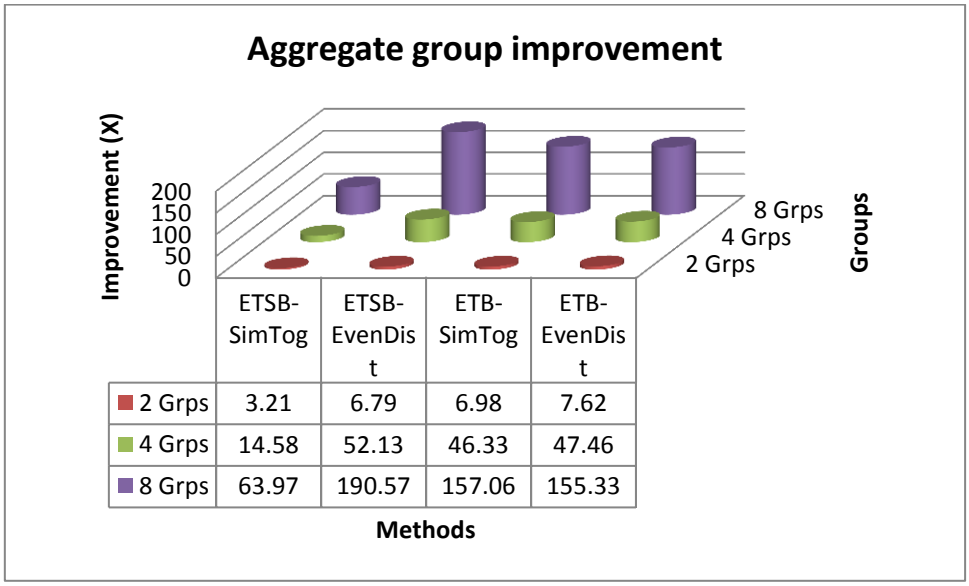


Figure 55: Percentage and mean scheduling time of the GPMS methods



	ETSB-SimTog	ETSB-EvenDist	ETB-SimTog	ETB-EvenDist
2 Grps	3.21	6.79	6.98	7.62
4 Grps	14.58	52.13	46.33	47.46
8 Grps	63.97	190.57	157.06	155.33

Figure 56: Aggregate group improvement

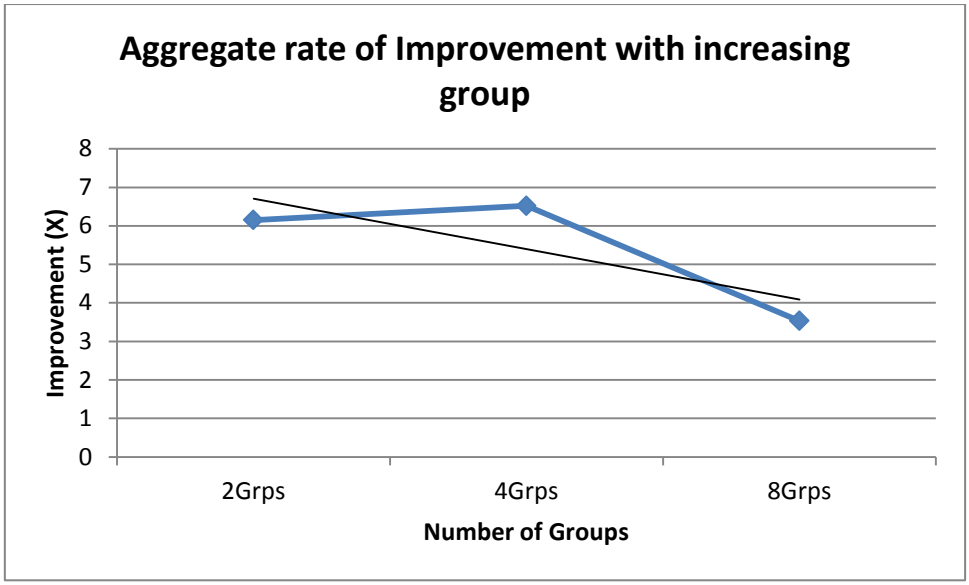


Figure 57: Aggregate rate of improvement with increasing group

5.6 Summary

This chapter has discussed the results, analysis and evaluation of all the methods against the ordinary MinMin and a comparative analysis between the GPMS methods. Three job grouping methods (Priority, ETB and ETSB) and two machines grouping methods (EvenDist and SimTog) where used; this gave a combination of six group scheduling methods: Priority-SimTog; Priority-EvenDist; ETB-SimTog; ETB-EvenDist; ETSB-SimTog; and ETSB-EvenDist. All grouping methods performed significantly better than the ordinary MinMin.

Also, some methods performed better than others. GPMS methods that ensure jobs are more equally shared (balanced) into groups performed better than other method that does not guarantee balancing of jobs. For instance, both the ETB and the ETSB performed better than the Priority method but the ETSB method performed better than the ETB method because the ETSB method balances the jobs even more fairly. Also, machine grouping methods that balances machines into groups (like EvenDist) also performed better than the (SimTog) method that does not share machines evenly into groups.

Increasing the number of groups also increases the performance improvement against the MinMin as more groups and parallel scheduling reduces the per instance scheduling time. However, within the same GPMS method, increasing the number of groups slowed the rate of

improvement between successive groups. This was due in part to the contention for resources that the increased number of threads introduced.

CHAPTER SIX

GENERAL DISCUSSION ON RESULTS AND OUTCOMES

CHAPTER SIX
GENERAL DISCUSSION ON RESULTS AND OUTCOMES

6.1 Introduction

This chapter presents a general discussion on the work and also discusses briefly the impact of shared resource contention among threads. This research recognizes that Grid computing is an important component in managing the data explosion currently affecting society. It also acknowledged that multicore computing technology is speedily pervading both the domestic and work lives. On this backdrop and given the fact that most current Grid scheduling algorithms are sequential, the task was to design a method that enables Grid schedulers harness the benefits of multicores in the scheduling task. Job and machine grouping methods were employed and several instances of independent and simultaneous scheduling (multi-scheduling) were simulated while threads were used for parallelization.

6.2 Overview of Approach and Results

This research introduced the GPMS which uses three grouping methods for scheduling Grid jobs in parallel. The methods are the Priority method, the ETB method and the ETSB method. Also two machine grouping methods, the EvenDist method and SimTog method, were introduced as part of the GPMS. The methods are designed to be used in batch scheduling and involve categorizing jobs into groups. Grid machines are also categorized into the same number of groups using the two methods. Job groups and machine groups are then paired and the MinMin scheduling algorithm is executed in parallel within the paired groups. Multiple threads were used to achieve parallel scheduling. The Priority method grouped jobs based on priority attributes while the ETB and ETSB methods employed the execution or processing times of the jobs for the grouping. Several experiments were performed. The Priority method used only four groups while the ETB and ETSB methods varied the groups between 2, 4, 8 and 16. The number of threads was also varied from 1 to 16 (in steps of power 2). Results show that by sharing jobs and machines into groups before scheduling, the pre-computation time for the algorithm and the scheduling time is drastically improved.

Users' jobs for the experiment were sourced from the Grid Workloads Archive (Anoep et al. 2007), while Grid sites, machines, CPUs and job execution times were simulated. The experiment was executed on one of the Coventry University's HPC's machine locally known as (Pluto).

6.3 Priority Method

The Priority method groups jobs based on priorities. Priorities were assigned to jobs based on the number of processors requested by the user on submission. The system was designed and implemented, tested, analysed and evaluated. Results and analysis shows that categorizing the jobs into four groups and scheduling the jobs in parallel reduces the total scheduling time by large margins. The correlation analysis showed that the relationship between GPMS methods is strong - this indicates that the results are reliable, not random and can be reproduced.

From the experiment results in Chapter Five, the MinMin algorithm used 242033ms to schedule a range of jobs from 1000 to 10000 in step 1000. The Priority-EvenDist method, took 35807ms to schedule the same range of jobs from 1000 to 10000 (step 1000) while the Priority-SimTog method took 41006ms to schedule 1000 to 10000 (step 1000) jobs. The Priority-EvenDist method recorded 6.76 times performance improvement over the ordinary MinMin algorithm which represents 85.21 % while the Priority-SimTog method performed better than the MinMin algorithm by 5.90 times representing 83.06%.

The results from the Priority method were better than the ordinary MinMin algorithms because grouping the jobs before scheduling in parallel reduced the number of per-instance processing by the algorithm. However, the growth pattern from the Priority method also tended towards polynomial as the number of jobs increases. That means the performance was degrading as the number of jobs increases which is expected since the MinMin scheduling time is polynomial. Inspection on the job input file reveals that more jobs were of low priority. Hence, jobs were not uniformly distributed into the four groups. Instead more jobs were sorted into the low priority group. This increased the per-instance scheduling time of the jobs in the group and impacted the efficiency. Thus, the effect of the grouping, which would have dampened the polynomial effect, was not achieved to its full potential.

From the results and analysis made, the conclusion is that the Priority method can be an effective way of reducing scheduling time. The splitting of jobs and machines into groups meant that the MinMin algorithm took less time in computing or estimating the completion time of jobs on all machines. The MinMin method is polynomial in nature, thus savings can be made through using smaller groups even without parallelisation. However running each grouped pair in parallel achieves still greater processing time benefits. Also, the nature of the input set and the machine grouping approach has an impact on the effectiveness of the method.

6.4 The ETB and ETSB Methods

The ETB and the ETSB methods were proposed to remedy the shortcomings inherent in the Priority method, which showed that the system might under-achieve parallelism due to the fact that more jobs could be sorted to a single group. Secondly, the number of groups in the previous Priority method was constant and it was not possible to determine with certainty about the effects of grouping jobs and machines. The ETB and the ETSB method were designed. With these two new methods, the number of groups can be varied, and jobs are not grouped based on priorities but rather by other methods which ensures uniformity in distribution amongst the groups. Both ETB and ETSB methods were executed in combination with the two machine grouping methods.

Execution Time Balanced (ETB)–this method estimates execution time of all jobs based on attributes and then balances or groups the jobs based on the execution times across groups.

Execution Time Sorted and Balanced (ETSB)–this method also estimates execution time of jobs. However, jobs are first sorted based on the execution times before balancing (grouping) them.

Results showed that grouping of jobs before scheduling increases the efficiency of the Grid scheduler by large margins and the efficiency increases with increase in the number of groups. Grouping jobs before executing the scheduling in parallel within the groups improved Grid scheduling algorithms performance by a range of 3.21 to 7.62 times when using two groups to schedule. With four groups, scheduling efficiency improved by a range of 14.58 to 52.13 times and when using eight groups, scheduling improved by a range of 63.97 to 190.58 times. Percentage-wise, these results showed that using two groups improved the scheduling

efficiency by 81% to 87% percent. Four groups improved the efficiency of scheduling by 97% to 98% while eight groups increased the performance by up to 99%. Between the groups, there was 80 to 84% improvement between four groups and two groups. Between eight and four groups, there was a 67% to 69% improvement.

Cumulatively, all 2 group methods made a combined mean of 6.15 times improvement over the ordinary MinMin. All 4 group methods made a combined mean of 40.13 times over the ordinary MinMin and all 8 group methods made a combined mean of 141.73 times improvement over the ordinary MinMin. Between the groups, 4 groups made an aggregate mean improvement of 6.5 over group 2 while 8 groups made an aggregate mean improvement of 3.5 over 4 groups. See Table 40.

Though there was improvement in speedup across the range of jobs by all methods, nevertheless, there was a pattern exhibited by the performance graph in all the cases. As the number of jobs increases, the speedup also increased up to a point then begins to level-off or decline. For each method and on each schedule, the speedup improves from the beginning (at 1000 jobs) to a point (say at 4000 or 5000 jobs) then declines for the rest of the period (up to 100000 jobs).

Likewise, there was a general decline in performance characterised by all methods. Though there was general performance improvement over the MinMin scheduling algorithm with increasing groups, this was not the case between two successive groups. Within a method, the rate of improvement was declining. Grouping of jobs therefore improves performance generally but within a method and between two successive groups, the improvement rate was marginal and declining. The general decline in performance between successive groups when using same method was also exhibited when the aggregate average performance for all the methods was examined. This characteristic can be partially attributed to overheads that results with increase in groups. Within the same GPMS method, the efficiency factor of the method is the same because they use the same scheduling strategy and the differences in performance is brought about by the differences in number of groups which also means number of threads as increased groups also increases the number of threads. As the number of groups increases, the number of threads used in scheduling also increases (one thread per group). This impacted the result as the threads contend for shared resources.

6.5 Differences between ETB and ETSB Methods

In demonstrating that manipulation of input jobs can be exploited in improving Grid scheduling, a comparative analysis was carried out between the ETB and ETSB methods vis-à-vis machine grouping methods. The ETB method performed similarly to the ETSB method when using the EvenDist machine grouping method because both machines and jobs were evenly distributed in this case. The ETB performed far better than the ETSB method when using SimTog to group machines.

6.6 Comparison of the ETB, ETSB and the Priority Methods

The GPMS methods include the Priority, the ETB and the ETSB methods. Both ETB and ETSB performed better than the Priority method because in the Priority method, jobs were not uniformly distributed based on priority attributes and therefore large number of jobs was assigned to the one machine group. Hence, scheduling from the group took relatively longer, increasing the overall scheduling time disproportionally due to the polynomial-time characteristics of the MinMin algorithm. If jobs were evenly distributed into the groups, the method would have performed relatively better compared to the other methods. Results from the experiment were near perfectly correlated and consistent (they were all tending to 1) – indicating that the results are reliable and can be reproduced. The standard deviation for all the methods except the grouping methods was close to the mean of the methods.

6.7 Comparison of Machine Grouping Methods (EvenDist and SimTog)

The ANOVA results in Table 36, Test 8 showed a difference between ETSB-EvenDist vs. ETSB-SimTog. This indicates that the methods employed in grouping machines effects the result differently when the job grouping method is the same. Generally, it was observed among all the three job grouping method that machine grouping methods that distribute machines fairly equally into groups like the EvenDist method performed better than the SimTog method that does not share machines evenly into groups.

6.8 Load Balancing in the GPMS

The research showed that the Priority method did not work well in the experiment because of poor load balancing. Hence the development of the ETB and ETSB methods for job grouping to ensure more even distribution of jobs according to estimated size. Of these two, the ETSB ensures better grouping of jobs according to job size. SimTog and EvenDist offer alternative methods of grouping machines. EvenDist provides the most balanced grouping of machines whereas SimTog groups similar machines together.

The experiment found that GPMS methods that ensure jobs are equally shared (balanced) into groups (like ETSB) performed better than other methods that does not guarantee balancing of jobs (like Priority) into groups. Also, machine grouping methods that balances machines into groups (like EvenDist) also performed better than the (SimTog) method that does not share machines evenly into groups. However the charateristics of the incoming jobs might determine the most suitable combination of job grouping and machine grouping method. The GPMS does not presently include a dynamic load balancing mechanism but the idea of dynamically employing different methods to handle differing job sets according to prevalent characteristics could be an extension to the system.

6.9 Impact of shared resource contention on the overall result

Shared resources are managed exclusively in hardware and most proposed solutions to avoid the shared resources contention require modifying the OS memory management subsystem or hardware (Liu et al. 2012). Meanwhile, rights and access to the use of the HPC on which the experiment was conducted was limited. Hence, the impact of shared resources contention was evident on the outcome of the results.

6.9.1 Impact of thread contention between the GPMS and MinMin

The effect of thread contention for shared resources impacted tremendously on the overall result and the efficiency of the GPMS method, the use of more threads (as a result of increased groups) to access the same source file introduced communication overheads and shared resource contention resulting in ineffective use of the caches and consequently led to increased cache-miss rate. The impact affected the overall improvement recorded by the

GPMS method over the ordinary MinMin. Most noticeably is when the number of threads increases as a result of increased group. This can be seen in Figures 28, Figure 29, Figure 40, Figure 41 and Figure 56. However, the impact of shared resource contention was more noticeable between successive GPMS groups than against the MinMin, this is because the GPMS method performed far more efficienctly compaired to the ordinary MinMin. Within the same GPMS method, the efficiency factor of the method is the same because they use the same scheduling strategy and the differences in performance is only determined by the differences in number of groups which also means number of threads as increased groups also increases the number of threads. Between the GPMS method and the MinMin, the efficiency factor is determined both by the method (or strategy) and the number of groups (or threads in this case). The comparisons were made at the point where both the GPMS and the ordinay MinMin used the same number of threads. The effect of resource contention is therefore not too noticeable between the GPMS methods and the ordinary MinMin because both methods used the same number of threads and the overall gains of the GPMS method (even with increased threads) far outweigh the impact of shared resource contention between it and the ordinary MinMin.

6.9.2 Impact of thread contention between successive groups within the GPMS method

The negating impact of shared resources contention was noticeable when using the same group method; as the number of groups increases, even though the performance of the successive group is better than that of the previous group, but the trend is negative. That is to say the rate of improvement between two successive groups within a given method was declining. This is because within the same GPMS methods, the same scheduling strategy is used and the difference in performance (or the efficiency factor) is a result of the differences in the number of groups used – which in this case is the same as the number of threads.

The groups used in this work are intended to increase parallelism in scheduling and since increased parallelism (more cores within a system) comes at a cost (shared resources contention), this affected the general performance of the method as the number of groups increases. As the number of groups increases, more threads are required to match the number of groups to carry out the parallel scheduling (one thread per group - two groups used two

threads to schedule, four groups used four threads and eight groups used eight threads to schedule). This created more contention between the (increased) threads for same resources. The negating impact was evident on the overall result and the analysis between successive groups.

6.9.3 Impact of thread contention on makespan in the GPMS

The focus of this research has been on the parallelisation of the scheduling activity rather than on makespan. However makespan is crucial and there would be little point in improving scheduling time if the resulting schedules made for longer makespan. The GPMS is intended to schedule independent jobs to the cores of the machines. Estimate makespans were calculated to give an outline assurance that the parallelisation of the scheduler achieved appropriate makespans but more detailed consideration of this and the effects of contention is needed. As the research literature exposed in section 2.3.3 has shown, the nature of the tasks themselves can affect contention as this is why deeper analysis is needed. As the jobs handled by the GPMS are independent there would be little contention over data access but there could be contention over the use of the LLCs, shared buses and DRAM controllers. The makespan currently calculated in this research does not include extra time for such contention as the concentration was on the multi-core aspect of the actual scheduling process. Delving deeper into the makespan aspect is a subject for future work.

6.10 Summary of Findings

The following were the findings made in this research:

- Grouping of jobs can improve scheduling efficiency and increase scheduling-throughput
- Increasing the number of job - machine groups directly increases the scheduling efficiency respectively.
- The idiosyncrasies of the input job set can have an effect on the scheduling outcome depending on the scheduling or grouping method used. This was evident with the Priority method where the attributes of the jobs were skewed and more jobs were sorted into one priority group.
- The attributes of the incoming job affect the quality of the resulting schedule.

- Increase in the number of groups (which also translate to increase in the number of threads) improved performance against the MinMin but the rate of improvement slowed between successive groups within the same method. This is because between the GPMS and the MinMin, the efficiency factor is determined by the strategy used for scheduling (grouping) and the increasing number of groups while within the same method, the scheduling strategy is the same and the impact factor or efficiency factor is determined only by the number of groups (or threads used). The slowing down of the rate of improvement is partially as a result of shared resource contention caused by increase in the number of threads as the number of group increases. Another reason is due to the polynomial nature of MinMin.

In conclusion, we say that the best results might be obtained by using an adaptive GPMS which can exploit different scheduling mechanisms depending on the characteristics of the incoming jobs. Future work will explore alternative grouping methods and how characteristics of input jobs can be harnessed such that appropriate grouping methods can be selected based on characteristics in an adaptive GPMS.

6.11 Summary

This chapter presented further discussions on the results and statistical analysis of the methods used in the experiments. It also brought together some key observations on characteristics exhibited by the various methods, together with some explanations.

The next chapter shall discuss the GPMS system in a different light relating it or comparing it to other established systems such as GridSim, gang scheduling, Condor and the DIANE scheduler.

CHAPTER SEVEN

COMPARISON OF GPMS AND PREVIOUS RESEARCH

CHAPTER SEVEN
COMPARISON OF GPMS AND PREVIOUS RESEARCH

7.1 Introduction

In this chapter a review is provided of the GPMS approach in comparison to previous research, some of which was introduced in Chapter Two. The GPMS is a simulator and whilst producing a simulator was not the primary aim of this research (rather the aim was to explore group based multi-scheduling methods) the implementation used in the exploration required simulation of Grid scheduling. Thus it has a relationship with previous work in simulation. Initially this chapter discusses simulation in Grid Systems and then compares the GPMS with a well-known Grid simulator, namely GridSim. Secondly the GPMS uses group scheduling and hence a later section of this chapter compares the GPMS to Group and Gang scheduling. Lastly it is interesting to compare the work with previous iconic distributed systems. The final section of the chapter compares the work to Condor and DIANE scheduler.

7.2 The Simulation Approach

The management and evaluation of resources and scheduling of applications in a heterogeneous environment where the resources are geographically distributed in multiple administrative domains managed and owned by different organizations, where different policies may be implemented is a complex challenge. Effectively evaluating the performance of scheduling algorithms in such environments requires that different scenarios be tested in a controllable and repeatable manner, like varying the number of resources, users, users' requirements and tasks. But this is difficult because resources in the Grid span across different administrative domains with varying policies, users, time zones and priorities. Moreover, many researchers do not have access to ready-to-use test bed infrastructure and cannot bear the burden of building such systems because of cost. More so, most existing test beds are limited in size and domains. Hence, testing and evaluating scheduling algorithms with such systems is difficult. This introduces a number of challenges in resource management and application scheduling the Grid.

Simulation and modelling has emerged as an important tool for modelling and evaluating real world systems/scenarios and many standard and application-specific tools and technologies have been developed and used extensively for modelling and evaluating real world scenarios. This has necessitated the development simulation languages e.g. Simscript (CACI), simulation environments e.g. Parsec (Bagrodia et al. 1998), simulation libraries e.g. SimJava (Howell and McNab 1998), and application specific simulators e.g. OMNet++ network simulator (Varga 2001). There also exist tools for simulating application scheduling in Grid computing environments. These include Bricks (Aida et al. 2000), MicroGrid (Song et al. 2000) Simgrid (Casanova 2001) and GridSim (Buyya and Murshed 2002) toolkit.

7.3 Some Grid Simulation Tools

This section discusses some simulation tools and technologies for simulating the Grid environments.

7.3.1 OptorSim

OptorSim is a package designed to imitate the structure of real Data Grid and investigate replica optimisation algorithms. It enables the studying of optimisation strategies under different conditions. In addition, it explores the stability and behaviour of different optimisation techniques (Bell et al. 2003). Written in Java, OptorSim was developed by the DataGRID (2004). OptorSim is simulated as a Grid with several sites, with each site having zero or more computational and data storage facilities. In OptorSim, computing elements run the jobs stored on storage elements and a resource broker controls the scheduling of jobs to computing elements. Optimisation in OptorSim is done in two phases: the first phase choses the computing element to run the job and the second phase involves the creation of replicas by the optimisation algorithm. This is aimed at achieving dynamic optimal replication during the running of the jobs. OptorSim uses two configuration files, one of the file descrbes the network topology while the other file comprises information about the logical names of files to be executed. OptorSim uses two types of optimisation algorithm, scheduling algorithms and replication algorithms. The replication algorithm creates geographically disparate but identical data sets aimed at reducing data access time and cost. OptorSim enable users to visualize the performance of a specific algorithm by providing a set of measurements which

can be used to quantify the effectiveness of the optimisation strategy under consideration, hence focusing on optimisation and data replication.

7.3.2 SimGrid

SimGrid (1999) is a toolkit created at the University of California, San Diego (UCSD). Implemented in C programming language, it provides core abstractions and functionalities that could be used to simulate specific distributed computing environments and to provide the tools for carrying out research in resource scheduling in distributed environments.

SimGrid simulation involves the creation of resources. Resources are created with two performance parameters, latency and service rates. These two parameters are used to simulate performance using a vector of time-stamped values or constants.

In 2003, SimGrid V2 was introduced with a new layer. This new layer provided the toolkit with the capability to model simulations in terms of communication agents with the capability of scheduling tasks on resources (Legrand, Marchal and Superieuredelyon 2003, and Casanova, Legrand and Quinson 2008).

In 2006, another model of SimGrid called Grid Reality and Simulation (GRAS) was deployed on top of SimGrid V2, this new model was to facilitate the operation of simulated codes in real time environments. The new model was built on top of the new software layer of V2; the Meta-SimGrid (MSG) in simulation mode and is built on top of the socket layer in real mode, introducing what is known as SimGrid V3 (Casanova, Legrand and Quinson 2008).

SimGrid is limited because of its restriction to a single scheduling entity and time shared system. Simulation of multiple users is difficult and the representation of resources or applications with separate policies and specifications is complex.

7.3.3 MicroGrid

MicroGrid (2004) is an online simulation tool designed for the Globus toolkit to model applications created in Globus to be carried out in a controlled environment. The package was developed in the University of California in San Diego (UCSD).

MicroGrid is designed to provide a platform that supports the simulated execution of real life applications. MicroGrid supports the running of applications that use dynamic resource allocations. It provides a mechanism for repeatable experiments in order to observe and study design aspects for applications and middleware, exploration of extreme circumstances and choices of application deployment, Grid resource allocation and network design.

MicroGrid uses a virtual Grid configuration file to build corresponding simulation objects required to create the virtual Grid. MicroGrid models applications and middleware to be executed on virtual machines in near real-time. Simulation objects in MicroGrid include network elements and computing resources.

Users of MicroGrid are first required to specify a set of virtual resources before specifying the physical resources to be used for the computation and online network simulation. Users are then be able to submit the application as a task on the virtual Grid, and observe the execution (Xia, Casanova and Chien 1999, Xin, Xia and Chien 2004, and Huang, Casanova and Chien 2006).

The limitation of MicroGrid is that the package is tied to the Globus toolkit which produces a significant amount of overhead. Moreover, using MicroGrid to model a large number of applications, environment and scenarios requires a significant amount of time.

7.3.4 GridSim

GridSim (Buyya and Murshed 2002) is designed to effectively simulate the Grid and evaluate applications in varying scenarios; it is a framework for deterministic modelling and simulation of resources and applications to evaluate scheduling strategies in the Grid. GridSim is java-based and has the capability to support modelling and simulation of heterogeneous Grid resources, users and applications.

The GridSim toolkit supports modelling and simulation of a wide range of heterogeneous resources, such as single processor or multiprocessors systems, shared and distributed memory machines such as PCs, workstations, and clusters with different capabilities and configurations. It can model application scheduling on various classes of parallel and distributed computing systems such as clusters, Grids and P2P networks.

GridSim is a very popular simulation tool used by researchers in simulating test scenarios and has proved to be generic, comprehensive and adaptable in various ways because it allows various scheduling algorithms to be simulated and evaluated.

GridSim has features that allow the modelling of heterogeneous resources. Resources can be modelled to operate under space- or time -shared mode. Time sharing ensures that threads are scheduled to execute on processors at time intervals. Space-sharing entails the scheduling of cores to execute completely the thread chosen to run, before executing the next.

Resource capability can be defined in the form of MIPS (Million Instruction per Second) as per the SPEC benchmark. Resources can be located in any time zone. Weekends and holidays can be mapped depending on resource's local time to model non-Grid (local) workload. Resources can be booked for advance reservation.

Applications with different parallel application models such as Clusters, Grids and P2P networks can be simulated. Application tasks can be heterogeneous, CPU intensive or I/O intensive. It supports simulation of both static and dynamic schedulers, any number of application jobs can be submitted to a resource.

Multiple users can submit tasks for execution in the same resource, which may be time - shared or space-shared. The network speed between resources and between users and resources can be simulated. And finally, statistics of all or selected operations can be recorded and analyzed using GridSim statistical analysis methods.

7.3.4.1 GridSim Entities

GridSim entities can be simulated as single processor, multiprocessor or heterogeneous resources that can be configured as time- or space-shared systems. Different time zones can be simulated to represent geographic distribution of resources. It can also simulate networks for communication among resources. GridSim also supports the creation of multi-threaded entities which run in parallel in their own threads.

User

In GridSim, Grid users are represented by a User entity; each user is represented by an instance of the User entity. Each User is distinguished from other Users by number of tasks to be submitted, execution time of each task, scheduling optimisation strategy (which could be Time, Cost or Cost/Time which also refers to Deadline, Budget or Deadline and Budget combined), task creation rate and Time Zone.

Resource Broker

Each User is connected to a resource broker; each resource broker is represented by a Resource Broker entity. Each user submits their tasks to the resource broker they are connected to, and the resource broker sends the tasks to the resources according to the Users optimisation strategy: Cost, Time or Cost/Time.

Resource

Each resource in GridSim is represented by an instance of the resource entity, a resource entity is a reusable entity that is deployed in the Grid and used to fulfil tasks submitted by Grid users. One resource entity differs from the other resource entity according to factors such as: the number of Machines in each resource; the number of Processing Elements (PEs) inside each Machine; the speed of each CPU or processor measured by MIPS; the cost of each processing unit; the resource allocation policy which is either time-shared allocation policy or space-shared allocation policy; local load factor; time zone where the resource is located; operating system; and system architecture.

Grid Information Service (GIS)

The Grid Information Service provides basic operational communication with users and resources in the GridSim package.

I/O Entities

I/O entities are represented by instances of the I/O entity. I/O entities enable the free flow of information between entities in GridSim. Each I/O entity is capable of executing in parallel in its own thread.

Gridlets

In GridSim, users' tasks are represented by Gridlet objects. Gridlets contain logical information about tasks, such as the size of the file, the user that originated the Gridlet, the

start time, finish time, total completion time, current status and the size of the file that is to be returned from the resource to the user.

7.3.4.2 Communication and Interaction between Entities

Interaction between entities in GridSim is done in the form of messages or events. These events are initiated by an entity to be delivered either with immediate effect or with a defined delay to other entities.

Internal Events are events that originated from the same entity while those that originated from external souces are called External Events. These events can be distinguished by the source identification associated with them. GridSim events are further classified into synchronous and asynchronous events depending on the service protocols.

7.3.4.3 Main GridSim Classes

The main GridSim classes in GridSim are:

GridSim: this class is responsible for initializing and starting the simulation. It also activates the simulation kernel and is required before any entity creation.

GridSimCore: this class is responsible in the management of I/O operations of an entity.

This class was an addition to the GridSim toolkit, aiming at taking over I/O operations: reducing the complexity of the GridSim class. Moreover, entities in this class are capable of knowing the bottleneck of a network route using the Gridsim.net package (Sulistio et al. 2007).

TrafficGenerator: this class generates the network traffic; it is used by entities of the GridSimCore class to determine bottlenecks of routes in a network topology.

Gridlet: This class is used for the creation of *Gridlets or u*sers tasks. The basic Gridlet class - before modification - contains information on the tasks submitted, including, task length and number of PEs.

GridUser: this class is used in the creation of *user* entities. It allows the users to communicate with and register with a GIS. It allows the user to query the GIS on resources availability.

GridResource: this class is used for the creation of different types of *Grid resource.*

AllocPolicy: this class is responsible in handling the internal resource allocation policy for a GridResource. The class allows the addition of new scheduling algorithms via extension of this class.

AdvancedReservation Classes: This class enables users to request for the use of resources in advance. Variations of the AR class includes: ARGridresource and ARPolicy.

These classes have added functionalities like: requesting reservations of PEs; creating reservations; committing reservations; modification of reservations; and reservation cancellation to GridSim.

7.3.4.4 GridSim Application Model

In the experiment, the application is modelled as a task farming application with 200 jobs. The jobs are packaged as Gridlets whose contents include the job length in MI (Million Instructions), the size of job input and output data in bytes along with various other execution related parameters. The job length is expressed in reference to the time it takes to run on a standard resource PE with SPEC/MIPS rating of 100. The processing time of Gridlets is estimated based on 100 time units with a variation of 0 to 10%. However, GridSim does not explicitly define any specific application model. The developer of schedulers and resource brokers defines them. The developers of GridSim experimented with a task-farming application model and believe that other parallel application models such as process parallelism, DAGs (Directed Acyclic Graphs), Divide and Conquer and other algorithms can also be modelled and simulated using GridSim.

7.3.4.5 GridSim Resource Model

In the GridSim experiment, resources were modelled as those of the WWG (World Wide Grid) testbed with different characteristics, configurations and capability. These configurations and characteristics reflect the latest CPU models. The processing capability of the PEs is modelled after the base value of SPEC CPU benchmark. The GridSim toolkit allows the creation of Processing Elements (PEs) with different speeds (measured in either MIPS or SPEC-like ratings). Machines are created with one or more PEs. Then, one or more machines are put together to make a Grid resource. The Grid resource can be a single

processor, shared memory multiprocessor (SMP), or a distributed memory cluster of computers.

Time-shared operating systems that uses round robin scheduling policy is used to manage the single PE or SMP type Grid resource while space-shared schedulers manages the distributed memory multiprocessing systems.

GridSim uses process oriented events to represent physical entities and simulates their behaviour. GridSim resources can send, receive, or schedule events to simulate the execution of jobs. Simulation of execution and allocation of PEs to Gridlet jobs are done using internal events. If there is a free PE when a job arrives, then space-shared systems start its execution immediately, otherwise, it is put in a queue. When a Gridlet job finishes execution, an internal event is generated to signify the completion of the Gridlet job. The PE allocated is then freed by the resource simulator and a check is made to determine if there are other jobs in the queue. If there are jobs waiting in the queue, then it selects a suitable job depending on the policy and assigns to the free PE.

7.3.4.6 Limitations of GridSim

GridSim is a generic simulation tool for the Grid and not tailored for some specific use. As is the case with most generic tools, it does not fully consider all the constraints in all circumstances and has to be adapted, modified or extended for specific use. This has necessitated the extension of the tool by several researchers such as Sulistio et al. (2007), Kalantari and Akbari 2009, Albodour, James and Yaacob (2010) and Qureshi, Rehman Manuel (2011) before use.

Albodour (2011) stated that GridSim only provides the basic and simple operations required to fully and accurately simulate a true Grid environment including the users, tasks and the scheduler. He stated that the creative flexibility of users, tasks and resources are limited. In GridSim, when users (called user entities) are created, they are immediately required to create Gridlets or tasks. In real Grid environment this is not typically the case as users are free to and should be able to create their tasks when they choose. He stated further that the Nimrod/G resource broker utilises a greedy method in satisfying users' requests without

taking into consideration any other requests from other users. He then argued that the greedy method does not consider load balancing or congestion in the Grid. Although GridSim can be adapted for the specific test or scenario, Albodour (2011) also argued that the Nimrod/G resource broker is limited in capability as it provides optimisation for budget and deadline scheduling only, when in reality, many other scheduling constraints are required.

Also, with GridSim, each independent task requires varying processing time and input files size which are created or defined through Gridlet objects which contain attributes related to the job and its execution details. A Gridlet object may contain information such as job length, disk I/O operations, the size of input and output files, and the job originator. These attributes help to determine the execution time of the job and the transportation time of the job.

But most jobs in the Grid Workload Archive used as source for the data do not contain these parameters required by GridSim. This influenced the decision to design a simulator that can work with the available parameters contained in the source file.

GridSim is a generic simulation tool and not tailored for specific use. As a result, it has to be adapted, modified or extended for specific use. Parallelisation could be simulated on GridSim if the parallel scheduler is broken down into parallel tasks and each task couched as a Gridlet but the actual scheduling code would not run, as GridSim does not support actual execution. Instead estimation would have to be made of the size of each scheduler task (or scheduler Gridlet) so that GridSim in turn could estimate the size of the parallel execution. Furthermore if the parallel scheduler was broken into tasks and input to GridSim for scheduling, there is no mechanism for adding the next level of simulation i.e. the task or payload scheduling of the input jobs. In other words, since GridSim does not support actual execution of tasks in the above described scenario, there is no way any output would be available to show the schedule that is determined by the scheduler and no facility to further simulate execution of that schedule. To carry out this investigation a new simulator which incorporated a parallel scheduler had to be written. It could have been possible to create the parallel scheduler and integrate it with GridSim in order to take advantage of some existing GridSim classes. However in this case the vast majority of the creation would have been new write rather than reuse because the functionality required does not currently exist in GridSim.

In summary the reasons for not developing the GPMS in GridSim were primarily so that it is not tied to a particular existing simulator, the lack of required functionality and to avoid potential constraints of developing within an existing product. Interesting future work could be the execution of the GPMS experiments using existing simulation tools like GridSim.

7.4 The GPMS Simulator

The simulator reads jobs from a file then calculates job sizes and job priorities. It then reads simulated machines from a machine file from simulated Grid sites. Based on the scheduling algorithm, it simulates job execution on machines and allocates jobs to machines.

The simulator is made up of the following packages and classes:

***Algorithms*:** This package contains the scheduling algorithm class. The scheduling algorithm class defines the scheduling or allocation policy to implement in the test. The algorithm used in this experiment is the MinMin algorithm but other scheduling algorithms can be developed and added to this package as a class then called in the SchedulingAlgorithmI class. This class therefore enables the simulation to be generalised or extended.

SchedulingAlgorithmI: this class calls the scheduling policy class and executes it. It takes a batch of jobs information, information about each Grid site and information about every machine in the Grid. It then produces for each machine in all Grids a list of jobs to be executed by that machine (in order). In other words, it simulates the allocation/execution of jobs to machines based on the scheduling algorithm or on the allocation policy.

***Entities*:** This package contains classes used in defining the components of the Grid and Grid jobs. These include:

GridInformation: this file contains information making up the Grid. In the simulation, the Grid is made up of categories determined by its network bandwidth and the type of machines constituting that Grid. There are categories A to D Grid sites. Categories are based on the configuration of the machines (speed of processors, number of cores and RAM size). Category A contains machines with less processing power and number of CPU cores. Category B contain machines with better configuration (based on speed, number of cores and RAM size) compared to group A. The machines in group C are better in configuration rating

compared to machines in group A and B while machines in group D contain machines with the best configuration in terms of speed of processors and number of cores.

Each Grid has a unique Grid id and is constituted by Grid machines made up of different number of cores with varying CPU speed and RAM sizes. Furthermore, each machine has a *machine id.*

JobInformation: the job information class contains information about users' jobs like the job id, job size and the priority of the job. The job sizes can be defined as Very Large, Large, Medium and Small.

JobPriority: This entity defines the priority of jobs. There are four different priorities namely; Very High, High, Medium and Low. Categorizations of job priority are based on the attributes of the jobs. GPMS uses the number of job processors requested by the user to estimate the priorities of jobs. This is different from the method used for estimating job sizes which is based on both the number of processors requested and / or the requested time or the average CPU time used (this value was not always available).

WorkerConfiguration: this entity contains the configurations or attributes of the machines making up the Grid. The attributes includes the machine id, the number of cores, the speed of CPU and the size of RAM.

***Files*:** This package contains classes that reads and stores files temporarily for the simulation/experiment. These include:

***GridLogReader*:** This is a class that reads the jobs from the job log, computes the jobs sizes and the priorities of the jobs based on the attributes. Among the attributes read are:

JobID, SubmitTime, WaitTime, RunTime, NProc, AverageCPUTimeUsed, UsedMemory, ReqNProcs, ReqTime, ReqMemory, UserId, GroupId, ExecutableId, QueueId, PartitionId, OrigSiteId, LastRunSiteId.

GridsInformationFile: This class reads the Grid machines available and stores them for the scheduling experiment.

***ScheduledJobsFile*:** This class reads the jobs file and machine file, and then keeps a log of the scheduled jobs and the cores in the machines they were allocated to for each round of scheduling.

***Threading*:** This package contains the class ThreadPool.java which creates a thread pool to be used for the multi-schedulingexperiment.

***Simulation*:** This package contains classes that simulate the Grid environment with machines and jobs and also simulates the scheduling of jobs to machines. It also simulates the execution of jobs on the machines. The simulation package contains the following classes:

***Execution Simulator*:** This class simulates the execution times of the jobs on the CPU cores in the machines they are allocated to. It takes as input the file containing machines list (machine id and specification), file with original (job) log information (job id, log entry containing job size, etc.) and file containing scheduled jobs (job id and machine id), and produces as output a table of job execution on machines with the following attributes; job id, job info, machine id, machine speed, waiting time, finish time, execution time. Machines are simulated to comprise varying number of cores. Jobs are allocated to the CPU cores and the execution times of the jobs are computed on the allocated cores.

This class also simulates the usage of the CPU cores and when the next CPU will be available for allocation to the next job.

The execution times of the jobs are simulated with the AverageCPUTimeUsed by the job (provided in the log entry) but for jobs without this value, the execution time is computed from the job size in reference to the speed of the allocated CPU core compared to that of a standard machine with a 1GB RAM and 1GHz. Algorithm for simulating the execution times of the jobs is shown in Table 42.

Table 42: Algorithm for simulating execution times

```
If (averageCPUTimeUsed == -1)
    baseTime = Job.Size  = ReqTime * ReqNProcs
OR baseTime = ReqTime if the number of processors is unknown
  else
   baseTime = averageCPUTimeUsed
  time = baseTime * 1000 / processor CPU speed
```

Each log entry in the source file contains (among others):

- ReqTime - expected execution time provided by the user
- ReqNProcs - expected number of processors, provided by the user
- RunTime - time when the job was started to the time when it finished
- AverageCPUTimeUsed - time actually used by the processor to execute the task

***Job Size*:** The simulation is based on real Grid data from the Grid workload archive. The simulator reads the jobs from a file. Based on the available parameters of the jobs, it estimates the job size with the requested time, or number of processors requested or both. Where both attributes are not available, then it uses the actual time it took the job to execute (which is represented by the value AverageCPUTimeUsed) as the job size. In the source data, one of the two or both of the two values (requested time, or number of processors requested) were always present. Hence, the algorithm does not evaluate to the third option that uses AverageCPUTimeUsed. If the size cannot be determined, then the log entry is ignored. Table 43 shows the algorithm to estimate the file size.

Table 43: Estimating the job size

```
If (ReqTime != -1 AND ReqNProcs != -1)
        Size = ReqTime * ReqNProcs
    else if(ReqTime != -1)
        Size = ReqNProcs
    else
        Size = AverageCPUTimeUsed
```

- For this simulation, if the size cannot be determined, then the log entry is ignored.

Create_Table: this class creates a table of scheduling times for the algorithm based on the number of jobs (jobs limit), method used, and or number of threads used.

Test_Scheduling: this class enables the scheduling algorithm to access Grid Jobs and the Grid machines and allow scheduling based on the scheduling algorithm's policy.

Test_Execution: this class enables the ExecutionSimulator class to execute. It estimates how long it will take to complete all jobs as scheduled by the algorithm.

Test_Parameters: the test parameter class sets out the experiment detail. It specifies the GPMS method to apply, number or range of jobs (jobs limit), the steps of jobs, the number of threads and the number of groups to use for the experiment. It also specifies where to read the input files from and where to save the measured scheduling results to.

Stats_Jobdistribution: This class counts the distribution of jobs used in the experiment based on priority.

Start.java: This is the main class that calls the execution to take place. It also ensures that scheduling results are written out and saved to the output file. Two output result files are generated from the simulation. These are:

***ResultStatistics*:** this file contains a general statistics of the scheduling times obtained by the scheduling methods in scheduling n jobs by the group method, number of threads used, number of groups used. A sample result statistics header file contains the algorithm used, the number of groups used, the machine grouping method used, the job grouping method used, the number of threads and number of groups used, job limit, time taken to schedule n jobs, execution time, core time, average core time, average machine time and machine standard deviation. These values tell how the each core performs in the scheduling experiment. A sample result statistics header file with some data is shown in the Table 44.

Table 44: Sample results statistics file

Algorithm	GroupsCc	WorkersG	JobsSplitI	InsideJob	ThreadsCc	JobsLimit	Schedulir	Executior	AvgCoreT	CoreTime	AvgMachi	MachineTimeStDev
MinMinAl	N/A	N/A	N/A	N/A	1	1000	805	315000	2007	12923.14	10889	17075.33
MinMinAl	N/A	N/A	N/A	N/A	1	2000	2970	1080900	6338	34748.39	33618	46578.15
MinMinAl	N/A	N/A	N/A	N/A	2	1000	762	315000	2007	12923.14	10889	17075.33
MinMinAl	N/A	N/A	N/A	N/A	2	2000	3170	1080900	6338	34748.39	33618	46578.15
MinMinAl	N/A	N/A	N/A	N/A	4	1000	738	315000	2007	12923.14	10889	17075.33
MinMinAl	N/A	N/A	N/A	N/A	4	2000	3230	1080900	6338	34748.39	33618	46578.15
Proposed.	2	ByPerforr	ByEstimat	MinMinA	2	1000	133	315000	2654	15074.21	13996	21697.54
Proposed.	2	ByPerforr	ByEstimat	MinMinA	2	2000	690	1080900	6270	34920.94	33598	50470.78
Proposed.	4	ByPerforr	ByEstimat	MinMinA	2	1000	41	315000	2396	14114.68	13000	20725.63
Proposed.	4	ByPerforr	ByEstimat	MinMinA	2	2000	125	1080900	6218	33253.18	35242	65352.93
Proposed.	4	ByPerforr	ByEstimat	MinMinA	4	1000	42	315000	2396	14114.68	13000	20725.63
Proposed.	4	ByPerforr	ByEstimat	MinMinA	4	2000	127	1080900	6218	33253.18	35242	65352.93
Proposed.	8	ByPerforr	ByEstimat	MinMinA	8	1000	19	315000	2366	13650.36	12940	20782.94
Proposed.	8	ByPerforr	ByEstimat	MinMinA	8	2000	33	1080900	6535	35502.32	36748	66988.48

ExecutionResults: This file contains the execution result of the jobs on the machine cores. It shows which cores in the machines jobs were allocated to, the waiting time, finish time and execution time of jobs on allocated machines. It also shows the job id, job size and job priorities. It also shows the machine id, CPU speed, RAM size and core of machines on which jobs were allocated and executed. A sample execution result file generated from MinMin algorithm executing 1000 jobs using 4 threads is shown in Table 45.

In Table 45, jobs 5506, 5507, 4243, 4244, 2345 and 2346 were allocated to machine 363 in three rounds of scheduling. Machine 363 is made up of two cores (core0 and core1) and its CPU speed is rated as 3500MHz (3.5GHz) and the RAM size of the machine is 2G. The allocations were made in three different schedules. In the first schedule, jobs 5506 and 5507 with size 3600 (categorized under low priority) were allocated to core0 and core1 of machine 363. The jobs waiting time were 0 and they both took 1028 milliseconds to execute.

In the second schedule, jobs 4243 and 4244 with sizes 14400 (and categorized under low priority) had their waiting times as 1028 milliseconds (the time it took the first set of jobs to execute). The execution time was 4114 milliseconds and the finish time was 5142 milliseconds (5142-1028 equals 4114) and in the third schedule, jobs 2345 and 2346 with size 86400 categorized as medium priority had their waiting time as 5142 milliseconds and were allocated to core0 and core1 respectively. Their finish time was 29827 Milliseconds and execution time was 24685 milliseconds.

The next sets of jobs were allocated to another machine with id 1480. Machine 1480 has four cores ranging from 0 to 3. Its CPU speed is 4000MHz (4GHZ) and RAM size 2G (this machine is faster than machine 363 and was utilized more in the scheduling). Machine 1480 was used for four rounds of scheduling. Four different jobs were allocated to each core in each round of scheduling. In the first set, jobs 2115, 2126, 2141 and 2168 were allocated to cores 0, 1, 2 and 3 respectively. In the second round of schedule; jobs 2211, 2251, 2252 and 2253 were allocated to cores 0, 1, 2 and 3 respectively.

From the table, it can be seen that for each machine, the smaller jobs were first allocated and executed before the higher jobs. For instance, machine 363 executed low, low, medium jobs in the three rounds of schedule and the jobs sizes were 3600, 14400 and 86400 respectively. Machine 1480 was allocated and executed low, medium, very high, very high order of jobs.

The job sizes were 900, 1800, 14400 and 86400 respectively. This is because the MinMin algorithm which favours smaller jobs was used in scheduling. See Table 45.

Table 45: Execution results file (ExecutionResults_MinMin_4_10000.txt)

Job ID	Machin	Core	WaitingTime	FinishTime	ExecutionTime	JobID	Size	Priority	MachineID	CPU Speed	CPU Core	RAM
5506	363	0	0	1028	1028	5506	3600	Low	363	3500	2	2000000
5507	363	1	0	1028	1028	5507	3600	Low	363	3500	2	2000000
4243	363	0	1028	5142	4114	4243	14400	Low	363	3500	2	2000000
4244	363	1	1028	5142	4114	4244	14400	Low	363	3500	2	2000000
2345	363	0	5142	29827	24685	2345	86400	Medium	363	3500	2	2000000
2346	363	1	5142	29827	24685	2346	86400	Medium	363	3500	2	2000000
2115	1480	0	0	900	900	2115	3600	Low	1480	4000	4	2000000
2126	1480	1	0	900	900	2126	3600	Low	1480	4000	4	2000000
2141	1480	2	0	900	900	2141	3600	Low	1480	4000	4	2000000
2168	1480	3	0	900	900	2168	3600	Low	1480	4000	4	2000000
2211	1480	0	900	2700	1800	2211	7200	Medium	1480	4000	4	2000000
2251	1480	1	900	2700	1800	2251	7200	Medium	1480	4000	4	2000000
2252	1480	2	900	2700	1800	2252	7200	Medium	1480	4000	4	2000000
2253	1480	3	900	2700	1800	2253	7200	Medium	1480	4000	4	2000000
2170	1480	0	2700	17100	14400	2170	57600	VeryHigh	1480	4000	4	2000000
2171	1480	1	2700	17100	14400	2171	57600	VeryHigh	1480	4000	4	2000000
2172	1480	2	2700	17100	14400	2172	57600	VeryHigh	1480	4000	4	2000000
2173	1480	3	2700	17100	14400	2173	57600	VeryHigh	1480	4000	4	2000000
5348	1480	0	17100	103500	86400	5348	345600	VeryHigh	1480	4000	4	2000000
5366	1480	1	17100	103500	86400	5366	345600	VeryHigh	1480	4000	4	2000000
5367	1480	2	17100	103500	86400	5367	345600	VeryHigh	1480	4000	4	2000000
6306	1480	3	17100	103500	86400	6306	345600	VeryHigh	1480	4000	4	2000000
2601	1494	0	0	900	900	2601	3600	Low	1494	4000	4	2000000
2602	1494	1	0	900	900	2602	3600	Low	1494	4000	4	2000000

7.5 Comparison between GridSim and the GPMS simulator

7.5.1 Application Model

Both GridSim and the GPMS do not explicitly define any specific application model. Both simulators allow the user to define and execute the algorithm of their choice. The developers of GridSim experimented with a task-farming application model while in the GPMS simulation; the MinMin scheduling algorithm was used.

Both GridSim and GPMS simulations allows users to define the scheduling algorithm for use. Hence they both have the capability to accept the file system used by the algorithm. For instance, GridSim, users (on creation) are required to define gridlets while the GPMS simulator accepts jobs from a batched file.

7.5.2 Resource Model

The GridSim toolkit allows the creation of Processing Elements (PEs) with different speeds (measured in either MIPS or SPEC-like ratings). It also allows the creation of varying machines with different number of PEs and scaling with more machines to form a Grid resource. The number of Grid resources can be changed easily – making it dynamic. The simulator used in this research simulates machines with different cores and different speed rated in GHz. A combination of different machines with varying cores and varying speed are specified to constitute a Grid.

GridSim models both Time-sharing and Space-sharing events while the GPMS simulator assigns jobs directly to the cores, hence it models only space-sharing events.

7.5.3 General Features

Both GridSim and the GPMS simulator are built with classes using the same programming language (java).

GridSim has the features to allow for the modelling of heterogeneous resources. Resource can be located in any time zone, weekends and holidays can be mapped, and resources can be booked for advance reservation. Heterogeneous tasks can be CPU or I/O intensive. There is no limit on the number of application jobs that can be submitted to a resource. Network speed between resources can be specified. It supports simulation of both static and dynamic schedulers. Statistics of all or selected operations can be recorded and they can be analyzed using GridSim statistics analysis methods.

Most of the features in GridSim are not available in the GPMS simulator developed for this experiment. For instance the simulator used in the experiment did not model differentiation between CPU or I/O intensive tasks, it did not also consider weekends, holidays, advance reservation and different time zones.

Some features common to both simulators are: support for static and dynamic schedulers (as users are allowed to define them); Network bandwidth of the Grid site; space-shared scheduling; and the number of jobs that can be submitted are not limited.

The GPMS is set up to support experimentation in parallelisation of the actual scheduler rather than parallelisation of regular jobs. Different algorithms can be used to schedule in parallel the batched groups of jobs. GridSim is not set up to experiment with parallelisation of the actual scheduler.

Overall, GridSim is more generic, extensive and has more features while the GPMS was specific as the design was focused on the task at hand. Despite the current restrictiveness of the GPMS simulator, there is room for expansion and standardization.

7.6 Relationship of the GPMS System to Gang Scheduling

This section provides further discussion of gang scheduling and then explains how this relates to the GPMS.

7.6.1 Gang Scheduling

The performance of multiprogramming systems degrades when a parallel application does not have all its interacting processes scheduled at the same time (Marinescu and Wang 1995). Gang scheduling (co-scheduling) was proposed to efficiently manage the scheduling of cooperating processes of a parallel application in a multiprogramming environment (Ousterhout 1982). Gang scheduling is the concept of scheduling at the same time only the active processes in a process group - a set of tasks is scheduled to execute simultaneously on a set of processors. The aim is to allow tasks to interact efficiently by using busy waiting, without the risk of waiting for a task that is not currently running. Without gang scheduling, tasks have to block in order to synchronize. This is because a process in execution that requires data (or input) in order to continue always blocks to wait for the input and continues execution after the input is supplied, thus suffering context switch overhead (Al-Saqabi, Sarwar and Saleh 1997, Wiseman and Feitelson 2003, Frachtenberg et al. 2001, Corbalan, Martorell, and Labarta 2001 and Karatza 2001).

Gang scheduling offers many advantages for job and system efficiency, the system can be better utilized by the scheduler's ability to pre-empt jobs in several ways. However, gang scheduling can incur a relatively high overhead due to the effect of the context switch on the computing nodes. This is caused by the resource sharing between multiple jobs and context switches between processes (Frachtenberg et al. 2001)

In gang scheduling, jobs are pre-empted and re-scheduled as a unit across all involved processors. The notion uses the analogy of a working set of memory pages to argue that a "working set" of processes should be co-scheduled for the application to make efficient progress (Ousterhout 1982). Gang scheduling provides an environment similar to a dedicated machine where all of a job's threads progress together, and at the same time allows resources to be shared. In particular, pre-emption is used to improve performance in the face of unknown runtimes. This prevents short jobs from being stuck in the queue (Feitelson, Rudolph and Schwiegelshohn 2004).

Gang-scheduling aims at optimal utility of the CPUs. To this end, gang scheduling is concerned with grouping of tasks into gangs that complement the optimal use of the CPUs. With gang-scheduling, useful results can be attained with proper coordination of tasks and processors.

7.6.2 Gang Scheduling and the GPMS

Both gang scheduling and the GPMS aim at achieving high scheduling-throughput by optimally utilising computer resources. In gang scheduling, multiple processes are selected for scheduling (time-sharing) and execution on processors (space-sharing) at the same time while the GPMS system groups and schedules independent (users) jobs onto the cores of a multicore system (space-sharing). Gang scheduling is aimed at efficiently scheduling dependent (cooperating) processes in a multiprogramming environment while the GPMS is aimed at enhancing scheduling of independent jobs in a multicore system. Gang scheduling targets the CPUs of a multiprocessor while the GPMS targets the cores in a multicore system. Gang scheduling targets dependent jobs (gangs are made based on dependent relationship between the processes) while the GPMS targets independent jobs (groups are made based on characteristics (attributes) of the jobs and not based on their dependencies).

The GPMS uses grouping to improve efficiency in scheduling of Grid jobs, it does so by allowing threads to execute scheduling algorithms independently within the groups. Jobs and machines distributed into a group are local to that group. Hence, the thread for that group performs the scheduling operation between jobs and machines local to the group alone. This allows n (where n = number of groups) instances of the scheduling algorithm to execute in parallel. The groups provide platforms for threads to execute independently, taking advantage of the multicores. It allows the jobs and machines in each group to be treated as a scheduling entity accessible to the thread. The GPMS therefore enhances scheduling-throughput by enabling jobs to be multi-scheduled.

In summary, Gang scheduling deals with the scheduling of a set of interdependent jobs whereas group scheduling in GPMS deals with the parallel scheduling of independent jobs. Hence, the concept of groups in GPMS is different to the concept of gangs in gang scheduling.

7.7 Comparison between the GPMS and Condor

This section provides a further discussion of Condor and explains how this previous work relates to the GPMS.

7.7.1 Condor

Condor is a high-throughput distributed batch computing system (Thain, Tannenbaum and Livny 2005) that utilises both dedicated and non-dedicated computers (Roy and Livny 2004 and Tannebaum et al. 2001). Condor provides resource management mechanism for job management, scheduling policy, priority scheme and resource monitoring, (Thain, Tannenbaum and Livny 2005). When jobs are submitted to Condor by users, Condor chooses when and where to run the jobs, monitors the jobs progress, and also informs users when execution is completed.

Condor also provides users with extra computing power by allowing them to submit jobs to non-dedicated computers; non-dedicated computers are computers that are only occasionally available for Condor to access, such computers are desktop computers belonging to other users or distant computers under private control (Tannenbaum et al. 2001). The policies and mechanisms employed in Condor enable the resource owners to control how their workstations are used as a HTC resource (Livny et al. 1997).

Some of the mechanisms employed by Condor are:
ClassAds - this enables Condor to pair resource requests and resource owners
Remote System Calls - this enables Condor to allocate resources across administrative domains.
Checkpointing – this is a mechanism that enables Condor to revoke resources that must be freed due to owners' constraints and to resume the application from where it left off on another resource.
Match-making – this is the means by which Resource Requests and Resource Owners that satisfy each other are identified and paired together.

There is no centralised job submission system in Condor; rather, each machine contains its own (local) job queue from where jobs are submitted from. According to Roy and Livny

(2004) "*when users submit jobs to Condor, they do not submit to global queues, as they would in many other batch systems, instead, Condor has a decentralized model where users submit to a local queue on their computer"*. Users may submit to a cluster (jobs submitted with a description file is referred to as a job cluster) from their own desktop machine or workstation (Tannenbaum, Wright, Miller, and Livny 2001).

Condor workstations have a daemon that detects user I/O and CPU activities. A job from the batch queue is assigned to a workstation that has been idle for two hours; this job will run until the daemon detects a keystroke, mouse motion, or high non-Condor CPU usage. When that happens, the job is revoked from the workstation and taken back to the batch queue. Furthermore, applications in Condor must be able to execute as a batch job. The applications are executed in the background and so are unable to perform interactive I/O operations. All I/O operations are redirected to a file on the user machine (Tannenbaum, Wright, Miller, and Livny 2001).

The Condor system is designed to maximize the utilization of workstations with as little interference as possible between jobs scheduled by the system and the activities of the owners of the workstations with a guarantee that jobs must complete (Litzkow, Livny and Mutka 1988). The system was initially aimed at balancing the under-utilisation of workstations owned by some individuals with the higher processing need of others whose workstations offer them less.

Condor identifies idle workstations and schedules jobs onto them, and when the owner of the workstation resumes activity on the system, Condor checkpoints the remote job running on the system and allows the user full control of his system. It then transfers the checkpointed job to another idle workstation and resumes it on another idle workstation - if and when available (Thain, Tannenbaum and Livny 2005).

The ability to access both dedicated and non-dedicated computers creates two major complexities with scheduling in Condor. First is the need to remove or pre-empt job(s) that were executing on an individual computer when owners reclaim their idle computers (CPUs) – this is called pre-emption. The second is the need to deal with the heterogeneity of computers available to Condor.

Pre-emption is carried out to meet the needs of owners, users, and administrators and to deal with unplanned outages. Condor pre-emption occurs for the following reasons: on behalf of users when better resources become available; on behalf of resource owners to ensure that the owner's policy on sharing is met; and on behalf of the system administrators to meet the efficiency of the entire Condor pool of computers. Computer owners will only allow their computers to run Condor jobs if Condor does not negatively impact their activities. Checkpointing and pre-emption is done to meet the need of the owners and also to prevent loss of work when the job resumes a new available computer (Raman, Livny and Solomon 1998).

7.7.2 The heterogeneity of computers available to Condor

Computers accessible to Condor are of different varieties in architectures, characteristics and performance and with varying policies. Heterogeneity complicates the scheduling problem in several ways. Different processors can have unequal processing capacities and hence an even distribution of work among the available processors will not usually result in correct load-balancing. Secondly, variations in architecture and instruction set among the available processors impose hard constraints on the choice of targets for scheduling decision (Al-Saqabi, Otto and Walpole 1994)

To provide the maximum amount of computational power to its users, there is the requirement for Condor to cope with this variety and handle the complexities. In order to deal effectively with this heterogeneity, Condor uses *matchmaking* to pair user's jobs with appropriate computers. Pairing of jobs and computers is determined by their description in the *ClassAd* (classified advertisements).

The job's requirements (in the job's ClassAd) are evaluated based on the machine's context and the machine's requirements (in the machine's ClassAd) are determined based on the job context. Both job and machine ClassAd must evaluate to true for a match to be made. The matchmaker informs both the user agent and the owner agent when a suitable match is found. The user and owner agent would then go ahead to claim the match independently of the matchmaker.

Users of Condor submit their jobs to a decentralized local queue on their computer and not to a global queue as they would in many other batch systems. The Condor processes on the computer would then interact with the Condor matchmaker and the computers that run the job. Interaction with the matchmaker is called matchmaking, and interaction with other computers is called claiming. Each computer in a Condor pool runs only a single job at a time, not multiple jobs – although, computers with multiple CPUs may run one job per CPU (Roy and Livny 2004).

Due to the advantages in Condor scheduling system Frey et al. (2002) proposed the Condor-G system to leverage the intra-domain resource management methods of Condor and the inter-domain resource management protocols of the Globus Toolkit. This is to allow users of the Grid to harness the multi-domain resources as if they all belong to one personal domain.

7.7.3 Gang Scheduling in Condor

This section discusses scheduling or matchmaking schemes in Condor that employed gangs, set or groups.

The matchmaking scheme in Condor allocates single jobs to single resources; this makes the scheme inadequate in some application domains that require several resources to execute a given task. To make Condor effective and adaptable in environments dominated by distributed management and distributed ownership, a mechanism is required to enable the aggregate matching of job and resources. To this end, Liu et al. (2002) implemented *set-matching*, the method enhanced Condor's ClassAd to allow both bilateral (single jobs to single resources) matchmaking and multilateral (several jobs to several resources) matchmaking activity to take place. Set-matching is limited in handling a heterogeneous mix of resources. Raman, Livny and Solomon (2003) also implemented a multilateral approach to machmaking in Condor job scheduling. Referred to as *Gangmatching*, the method improved Condor's capabilities by extending ClassAd to enable multiple resources to be marshalled. Gangmathing uses a docking paradigm to group a gang of ClassAds with similar attributes with a machine operation. Another work in this direction is *Redline* implemented by Liu and Foster (2004), Redline is a symmetric matchmaking scheme that extended Condor's ClassAd and allowed for multiple matches to be made. Redline uses a very complex language for advertisement.

7.7.4 GPMS and Condor Comparison

Although there are similarities between the GPMS scheduler and Condor, it is clear that Condor, which has been developed over time, has more features for appropriate management of resources and is a more tried and tested system. However it is interesting to compare the systems considering a variety of aspects.

Goals

To provide users with the amount of processing power they require, available resources need to be optimally utilized; this calls for parallel execution of jobs on available resources. Recent trends in the cost/performance ratio of computer hardware have meant that the control of powerful computing resources is now in the hands of individuals and groups with a growing need of users who are throughput-oriented. Exploiting these resources to the benefit of the user is the goal of both Condor and the GPMS system. Both Condor and the GPMS system satisfy the computing needs of the throughput-oriented users by exploiting available resources for the simultaneous execution of users' jobs.

Distributed network of computers and distributed ownership of computing resources

Both Condor and the Grid seek to harnesses the computing power of a distributed set of computers on a network and controlled by different owner policies. Both the Grid and Condor provide a HTC environment intended to address the challenges introduced by distributed ownership of computing resources, allow users to transparently exploit the capacity of thousands of workstations simultaneously and properly manage the resources to offer high degree of parallelism. Condor exploits the processing power of several workstations from several owners with varying control mechanisms. The GPMS is also designed to exploit the dynamic and heterogeneous resources of the Grids. It does this by exploiting parallel multi-scheduling methods and exploiting the capacity of Grid resources for parallel execution.

Scheduling

Scheduling with the GPMS is in a way similar to Condor in that they both deal with batch jobs. The main difference between GPMS and Condor is that GPMS uses a parallel scheduler whereas Condor does not. Also, the focus on Condor is a broad approach to scheduling using dedicated and non-dedicated resources whereas GPMS uses just dedicated resources. Condor is much more developed and does things GPMS does not currently do but theoretically GPMS could be developed to do such things.

The Condor system is decentralised as jobs are submitted to a local queue on the user's computer. When jobs are submitted to Condor, a special file is generated containing arguments that help Condor create a ClassAd for the job which in turn helps Condor work towards running it on contributing resources. Grid jobs (used in the GPMS) are submitted to the central scheduler from where they are batched before scheduling.

Improvements to Condor system now enables the system to execute both bilateral (single jobs to single resources) matchmaking and multilateral (several jobs to several resources) matchmaking activities. The multilateral matchmaking capability of Condor is synonymous to the GPMS's multi-scheduling capability.

Matchmaking and job-machine pairing

The matchmaking used in Condor is synonymous with the job-machine pairing (used in the MinMin) done before jobs are despatched to machines in the GPMS. Condor executes one job at a time or one job per CPU, while the GPMS executes one job per core; meaning that one processor with multiple cores can execute several jobs.

The GPMS system takes all the machines in the Grid as one dedicated system while Condor has the capability to differentiate between dedicated and non-dedicated system and hence has the capability to manage them differently.

The idea of ClassAds (which represents the interaction of users jobs and owners machines) is also represented in the GPMS by the interaction between job groups and machine groups.

The attributes of jobs and configuration of machines used by the GPMS system for grouping purposes before implementing the scheduling algorithm is synonymous with matchmaking made with classAds in Condor.

Claiming in Condor which happens when a job's ClassAd and a machine's ClassAd are matched by the matchmaker can be likened to the process of allocation and despatch of a job to a processor's core by the scheduling algorithm in GPMS.

Pre-empting and migrating

Though not implemented in the GPMS, the Grid could be expanded to migrate jobs from one failed system to the other just like Condor would pre-empt, checkpoint and resume jobs from a reclaimed, failed or less powerful system to another system.

Leverage (which is a job's ratio of capacity consumed remotely to capacity consumed locally to support remote execution) is not required in the GPMS system because once users submit their jobs, the Grid scheduler does not require the user's local machine to perform any more tasks rather than receive the processed job after execution.

Checkpointing (which is the saving of the state of an executing task from a reclaimed remote machine) used in Condor is not used in the GPMS because the system assumes a dedicated set of Grid resources for its use.

Parallelism and increased throughput

A Condor pool can be viewed as a private computational Grid of desktop workstations that are managed for HTC use, a Condor system enables one job to execute on one CPU. Condor systems with several CPUs are able to execute several jobs; this is aimed at achieving high throughput, exploiting available resources to optimum, and enabling parallelism.

The GPMS achieves scheduling-throughput by exploiting multiple threads to simultaneously schedule independent groups in parallel on a HTC system. Also, scheduling on the GPMS system targets the cores of the machines, this enables several independent jobs to be executed on the cores in parallel.

File system

Both Condor and the GPMS uses batch systems to service users' jobs and owners/Grid resources. Batch systems are equipped with queuing mechanisms, scheduling policies, priority schemes, and resource classifications. Batch systems have been extended to deal with large multiprocessor, multicore computers and clusters of workstations and its policies have also been adapted to meet the needs of workloads that consist of both sequential and parallel applications (Livny and Raman 1999).

7.8 Relationship to DIANE

This section considers DIANE (Distributed Analysis Environment for GRID-enabled Simulation and Analysis of Physics Data) and discusses how the GPMS relates to DIANE.

7.8.1 DIANE

DIANE (Moscicki 2003) is a workflow management package for distributed master-worker applications that is built on top of the GRID middleware to provide high-level mechanisms for distributed application development and deployment. It interfaces semi interactive parallel applications with distributed GRID technology. DIANE provides high-level facilities and mechanisms for developing and deploying distributed applications with ease. Application developers are shielded from coding the communication mechanisms explicitly. Rather, they only implement the callback interfaces and describe the contents of input and output data messages, then, DIANE takes care of workflow management and message passing. The system is flexible, easy to configure, adaptable and scalable according to changing needs. It is language-neutral and it insulates the applications from the details of underlying middleware. DIANE is also interoperable.

The master-worker computing paradgm used in DIANE entails that client's jobs are sent to the Planner which then partitions the jobs into smaller tasks and allocates to the Workers for execution. There is also the Integrator which merges the results of execution and sends the final results back to the client. There is also the DIANE Master-Worker container which serves as host to the Integrator, Planner and Worker App and also provides the run-time context and set-up the environment.

7.8.2 Comparison between DIANE and the GPMS system

DIANE uses a Master-Worker model; the Master-Worker model employed by DIANE encourages partitioning of tasks by the master and assigning to workers to execute in parallel. Diane handles jobs which contain inter-dependent tasks while the GPMS system targets independent jobs. The unique feature of the GPMS is that it incorporates parallelism at the scheduler level as well as the execution stage, while DIANE focuses on parallelsm at the task execution stage only. At the execution stage, GPMS tasks are assigned to individual cores for independent execution.

With DIANE, jobs are sent to the planner which partitions the jobs into smaller tasks for execution. In the GPMS, independent users jobs are not partitioned but grouped for parallel scheduling onto the cores of Grid resources.

With DIANE, the application is shielded from the specific details of underlying middleware, thus making it easy for the user to configure, adapt and extend according their need. The GPMS system is also easy to adapt; the developer only has to define and execute the algorithm of their choice in the specific class and GPMS system will carry out the task of grouping jobs and scheduling the jobs in parallel.

The DIANE scheduler is more adapted for real-time and interactive distributed systems and has been applied for real-life use-cases in the domain of Distributed Simulation for Medical Physics and Space Science Applications. DIANE was used to perform a sizeable fraction of an *in silico* drug discovery application using the EGEE and other Grid infrastructures. At the ITU's Regional Radiocommunication Conference initiated by CERN, DIANE was successfully used to process large-scale data processing activities. The GPMS system targets scheduling of independent executable jobs and uses batch processing rather than real-time and interactive systems. The GPMS system has not been applied and tested as extensively as the DIANE.

7.9 Summary

This chapter discussed the GPMS approach in relation to other established systems. First, it discussed other Grid simulation tools. It then focused on GridSim and compared the GPMS simulation used in this research to GridSim. It then discussed gang scheduling and how the GPMS relates to gang scheduling. The discussion then shifted to Condor and how it compares to the GPMS. Lastly, the chapter discussed the DIANE scheduler and also made comparison between the DIANE scheduler and the GPMS.

The next chapter shall discuss contributions made to knowledge, draw conclusions and discuss future work.

CHAPTER EIGHT

CONCLUSION AND FUTURE THOUGHTS

CHAPTER EIGHT
CONCLUSION AND FUTURE THOUGHTS

8.1 Introduction

This chapter serves to bring the work to a close. It highlights the key points and outlines the contributions made to knowledge. Then it draws conclusions and discusses future work.

8.2 Contributions to Knowledge

This work is chiefly about taking advantage of multicore technology and parallelising the Grid scheduling task. The interest of most researchers in Grid scheduling has been on creating schedules such that overall makespan is decreased. This research improves on those efforts by providing a method by which the scheduling can also be carried out in parallel. This work has thrown new light into novel methods of exploiting parallelism to improve the efficiency of Grid scheduling algorithms. Job grouping and machine grouping methods were employed to improve the efficiency of Grid scheduling algorithms on multicore systems. The method took advantage of the underlying multicore for parallelism rather than leaving it in the hands of the system alone.

The contribution of this work has been on how to use grouping methods to harness parallelism in multicores and improve scheduling efficiency. The resulting Group Parallel Multi-scheduler (GPMS) can be used in any environment in which there is a requirement to schedule a batch of jobs onto a set of limited or available resources. Typical environments which could benefit are Grid and Cloud environments. Given the trend in these computing paradigms, the research has potential to be exploited widely.

The following are the contributions made to knowledge:

- The development of the grouping idea to support parallelization of Grid scheduling algorithms. Various methods of grouping were explored
- A Group-based Parallel Multi-scheduler (GPMS) was designed and developed. The GPMS included innovative methods to group jobs and machines:
  - The Priority method grouped jobs based on priority. Priority could be specified by users or estimated via job characteristics.

- The Estimated Time Balanced (ETB) method and the Estimated Time Sorted and Balanced (ETSB) method which ensure even distribution of jobs across groups were developed as enhanced methods to the Priority method.
- Two methods of machine grouping were introduced; Similar Together (SimTog) method and Evenly Distributed (EvenDist) method. These methods serve to support the job grouping methods so that groups of jobs and machines can be matched, thereby enabling parallel instances of the scheduling tasks.

This research aims to address the issue of Grid scheduling by employing a dynamic approach that exploits the gains of parallelism on multicores. In relation to the aims and objectives introduced in Chapter 1, it is safe to conclude that:

- The GPMS method can be an effective way of reducing scheduling time and improving scheduling in general. The splitting of jobs into groups and scheduling independently means that fewer read accesses are made on jobs and machines in each group. This reduces the scheduling time of the scheduling algorithms.
- Grouping of jobs can be adopted to harness parallelism on multicore machines to increase scheduling-throughput and improve scheduling efficiency. The MinMin method used in the test is polynomial in nature, thus savings can be made through using smaller groups even without parallelization. Furthermore, running each grouped pair in parallel achieves greater processing time benefits. Also of note is that the nature of the input set and machine grouping approach has an impact on the effectiveness of the method.
- Grouping of jobs before scheduling, in general, can reduce scheduling time and increase scheduling-throughput.
- Grouping of jobs before scheduling enhances parallelism by providing a platform for threads to execute independently.
- Grid jobs can benefit more from parallelism if grouping methods for both jobs and machines are exploited.

The bottom line is that for software applications to gain from the immediate benefits of multicore systems, concerted effort should be made to move both new and legacy applications towards parallelism. Grid scheduling will be better leveraged if this method of targeting multicores is adopted.

Let us consider again the research question introduced in Chapter One.

How can multi-scheduling and parallelism be exploited to take advantage of multicores in order to improve the Grid scheduling task?

This research has answered the above question in demonstrating the use of a Group-based Parallel Multi-scheduler (GPMS) which exploited grouping methods and parallelism to yield significant improve in performance over serial schedulers.

8.3 Conclusion

This work explored job grouping methods in a bid to increase throughput in scheduling Grid jobs by exploiting the multicore hardware. This informed the development of the GPMS method which used three different methods, Priority, Estimated Time Balanced (ETB) and Estimated Time Sorted and Balanced (ETSB) to group jobs. All methods used groups to create an independent separation so scheduling can be done in parallel and simultaneously from the independent groups. Two machine grouping methods: *Similar Together (SimTog)* and *Evenly_Distributed (EvenDist)* were used to group machines. Parallelism in scheduling was achieved using dynamic threads and by matching job groups with machine groups and scheduling paired groups simultaneously. The MinMin scheduling algorithm was used as the *insidegroups* scheduling method.

All methods achieved significant speedup and improved scheduling efficiency when compared to the ordinary MinMin. However, some methods achieved better performance improvement than other methods due to the characteristics of the jobs or machines which affected the grouping outcome. Thus we can conclude that the best results might be obtained by using an adaptive GPMS which can exploit the different scheduling mechanisms or algorithms depending on the characteristics of the incoming jobs and available machines.

8.4 Future Thoughts

Since the interest of most researchers in Grid scheduling has been on the scheduling of parallel independent jobs instead of parallelising the scheduling task, this research can open a new area of parallelisation of the scheduler; the parallel scheduling of parallel tasks, where all the interacting units of jobs or sub jobs are selected for parallel scheduling onto cooperating computer systems in parallel.

This work did not directly control the number of CPUs on the HPC in the experiment. Hence, the relationship between increased CPU and groups in relation to scheduling efficiency cannot be ascertained. Future investigation should seek direct control of the system on which the scheduler runs. This will ensure that the number of CPUs on the HPC or system on which the experiment shall be executed can also be varied. This will allow for the relationship between increased groups and increased CPUs or cores to be investigated.

This research showed that the characteristics of incoming jobs affected the performance of the grouping methods. Future work will explore alternative grouping methods and how characteristics of incoming jobs can be identified early and exploited such that appropriate grouping method can be selected based on job characteristics in an adaptive GPMS.

Furthermore, patterns of previous usage and performance could be collected and exploited to devise a method of determining the number of groups and threads for a particular job set. A future investigation would be to explore how dynamic and batch scheduling could be efficiently combined. At present the GPMS only uses batch scheduling.

The makespan currently calculated in this research does not include extra time for shared resource contention as the concentration was on the multi-core aspect of the actual scheduling process. Future work should explore how makespan is affected by shared resource contention.

Within the same GPMS method, increase in the number of groups (which also translates to increase in the number of threads) slowed the rate of improvement between the successive groups partially due to the impact of shared resource contention among threads. Further investigation should involve methods to reduce the impact of shared resource contention between threads.

In a complex environment, this study can be extended to include the implementation of multiple scheduling algorithms across the discrete job-machine groups. In that way, we can independently execute a mix of different scheduling algorithms on each of the independent groups. This will enable the use of suitable scheduling algorithms favourable to jobs in a particular group and the use of other scheduling algorithm favourable to other jobs in other groups. If characteristics or attributes of certain jobs do affect the schedulers efficiency, then this proposed method will provide the opportunity to exploit the benefits of one scheduling

algorithm (from one set of jobs in one group) against the disadvantages of the other (in another set of jobs in another group). This will enable implementation of a scheduling algorithm within a group based on which scheduling algorithm favours jobs in that group. When the implementation of different or several scheduling algorithms from different groups is finally achieved, such systems or schedulers shall be referred to as hetero-multi-scheduling systems while systems that implement one scheduling algorithm on multiple group of jobs (like the method presented in this work) can be referred to as a mono-multi-scheduling systems.

Lastly, the experiment was executed in a simulated environment and not on a real test bed. While the differences of a simulated environment and that of a real system or test bed are out of the scope of this work, it will be worthwhile to state here that effort should be made to test the experiment on a real test bed to ascertain the real functionality of the method.

References

References

Abraham, A., Buyya, R., and Nath, B. (eds.) (2000*)'*Nature's heuristics for scheduling jobs on computational grids*'.* in *Proceedings of the 8th IEEE international conference on advanced computing and communications (ADCOM 2000)*, 45-52

Abraham, A., Liu, H., Grosan, C., and Xhafa, F. (2008) 'Nature inspired meta-heuristics for grid scheduling: single and multi-objective optimization approaches'. in *Metaheuristics for Scheduling in Distributed Computing Environments* 247-272 Springer Berlin Heidelberg

Abraham, G, T., James, A., and Yaacob, N. (2015a) 'Priority-Grouping Method for Parallel Multi-Scheduling in Grid'. *Journal of Computer and System Sciences (81)6, 943-57* DOI: http://dx.doi.org/10.1016/j.jcss.2014.12.009

Abraham, G, T., James, A., and Yaacob, N. (2015b) 'Group-based Parallel Multi-scheduler for Grid Computing.' *Future Generation Computer Systems 50, 140-153* DOI: http://dx.doi.org/10.1016/j.future.2015.01.012

Adams, J, C., Ernst, D, J., Murphy, T., and Ortiz, A. (2010) 'Multicore Education: Pieces of the Parallel Puzzle'. in *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, ACM, 194-195

Agarwal, A., and Kumar, P. (2011) 'Multidimensional QoS Oriented Task Scheduling in Grid Environments'. *International Journal of Grid Computing and Applications* 2 (1), 28-37

Ahmad, I., and Kwok, Y, K. (1995) 'A parallel approach for multiprocessor scheduling*'. in Proceeding of the 9th International Symposium on Parallel Processing* 289-293, IEEE

Ahuja, S., Curriero, N., and Gelernter, D. (1986) 'Linda and Friends'. *Computer* 19 (8)

Aida, K., Takefusa, A., Nakada, H., Matsuoka, S., Sekiguchi, S., and Nagashima, U. (2000) 'Performance evaluation model for scheduling in a global computing system'. *International Journal of High Performance Computing Applications* 14(3), Sage Publications, USA, 2000.

Alan, K. (2006) *Parallel Java: an API for Developing Parallel Programs in 100% Java* [online] available from <http://www.cs.rit.edu/ark/lectures/pj03/notes.shtml> [18/06/2014]

Albodour, R., James, A., and Yaacob, N. (2010) *An extension of GridSim for quality of service.* 'The 14th International Conference on Cooperative Work in Design', held 14-16 April in Shanghai, China. ISBN: 978-1-4244-6763-1, 361 - 366

Albodour, R. (2011) '*A Flexible Model Supporting QoS and Reallocation for Grid Applications*.' dissertation. Coventry University

Albodour, R., James, A., and Yaacob, N. (2014) 'QoS within Business Grid Quality of Service (BGQoS)'. *Future Generation Computer Systems*

Albodour, R., James, A., and Yaacob, N. (2012) 'High Level QoS-Driven Model for Grid Applications in a Simulated Environment'. Future Generation Computer Systems 28 (7), 1133-1144

Alem, A, W, M., and Feitelson, D, G. (2001) 'Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling.' in *Parallel and Distributed Systems, IEEE Transactions on*, *12*(6), 529-543

Ali, S., Siegel, H, J., Maheswaran, M., and Hensgen, D. (2000) 'Task execution time modeling for heterogeneous computing systems.' *Heterogeneous Computing Workshop, 2000. (HCW 2000) Proceedings. 9th* (pp. 185-199). IEEE

Allcock, W., Bester, J., Bresnahan, J., Chervenak, A., Liming, L., and Tuecke, S. (2003) 'GridFTP: Protocol Extensions to FTP for the Grid'*. Global Grid Forum GFD-RP* 20

Allcock, B., Bester, J., Bresnahan, J., Chervenak, A, L., Foster, I., Kesselman, C., and Tuecke, S. (2002) Data management and transfer in high-performance computational grid environments. *Parallel Computing*, *28*(5), 749-771

Allcock, W., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raicu, I., and Foster, I. (2005) 'The Globus stripped GridFTP framework and server'. in *Proceedings of the 2005 ACM/IEEE conference on supercomputing.* IEEE

Al-Saqabi, K,H., Otto, S,W., Walpole, J. (1994) Gang scheduling in heterogeneous distributed systems, Technical Report CSE-94-023, OGI of Science and Technology

Amdahl, M, G. (ed.) (1967) 'Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities*'.* in *Proceedings of AFIPS Spring Joint Computer Conference*. held April at Washington D.C. Washington D.C: Thompson30483-485

Amudha, T., and Dhivyaprabha, T, T. (2011) 'Qos priority based scheduling algorithm and proposed framework for task scheduling in a grid environment'. In *International Conference on Recent Trends in Information Technology (ICRTIT),* 650-655 IEEE

Anoep, S., Dumitrescu, C., Epema, D., Iosup, A., Jan, M., Li, H., and Wolters, L. *(2007) The Grid Workloads Archive* [online] available from <http://gwa.ewi.tudelft.nl/dataset/> [27th June 2013]

Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., and Yelick, K. (2009) 'A view of the Parallel Computing Landscape'*. Communications of the ACM* 52 (10)*,* 56-67

Bader, D, A., Kanade, V., and Madduri, K. (eds.) (2007) *Parallel and Distributed Processing Symposium IPDPS 2007.* 'SWARM: A Parallel Programming Framework for Multicore Processors.' IEEE International

Bader, D, A., and Cong, G. (2011) 'SWARM: A Parallel Programming Framework for Multicore Processors'. in Encyclopaedia of Parallel Computing. ed. by Anon: Springer, 1966-1971

Bagrodia, R., Meyer, R., Takai, M., Chen, Y., Zeng, X., Martin, J., Park, B., and Song, H. (1998) *Parsec: A Parallel simulation environment for complex systems,* IEEE 31(10)

Bak, S., Yao, G., Pellizzoni, R., and Caccamo, M. (2012) 'Memory-Aware Scheduling of Multicore Task Sets for Real-Time Systems. In the *18th International Conference on Embedded and Real-Time Computing Systems and Applications, 2012,* 300-309, IEEE

Barney, B. (02/14/2012) *Introduction to Parallel Computing* [online] available from <https://computing.llnl.gov/tutorials/parallel_comp/> [02/22 2012]

Beav, I, D., Meleis, W, M., and Eichenberger, A. (2000) '*Lower bounds on precedence-constrained scheduling for parallel machines'.* in proceedings of the 29th International Conference on Parallel Processing. 549-553

Bell, W,H., Cameron, D, G., Millar, A, P., Capozza, L., Sttockinger, K., and Zini, F. (2003) 'OptorSim: A grid simulator for studying dynamic data replication strategies'. *International Journal of High Performance Computing Applications, 17(4) 403-416*

Bell, G. (2008) 'Bell's Law for the Birth and Death of Computer Classes '*. Communications of the ACM* 51 (1), 86-94

Berenbrink, P., Friedetzky, T., and Goldberg, L, A. (2003) 'The natural work-stealing algorithm is stable.' *SIAM Journal on Computing* 32(5) 1260-1279

Blumofe, R, D., and Leiserson, C, E. (1999) 'Scheduling multithreaded computations by work stealing.' *Journal of the ACM (*46) 5 720–748

Bolondi, M., and Bondanza, M. (1993) '*Parallelizzazione di un algoritmo per la risoluzione del problema del commesso viaggiatore'*. Unpublished master's thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy

Boloni, L., and Marinescu, D, C. (2000) 'An object-oriented framework for building collaborative network agents'. In *Intelligent systems and interfaces* 31-64 Springer US

Bondhugula, U., Baskaran, M., Hartono, A., Krishnamoorthy, S., Ramanujam, J., Rountev, A., and Sadayappan, P, (2008) 'Towards effective automatic parallelization for multicore systems'. In *International Symposium on Parallel and Distributed Processing,* 1-5 IEEE

Borthakur, D. (2007) 'The Hadoop distributed file system: Architecture and design'. Hadoop Project website *11*(2007), 21

Braun, T, D., Siegel, H, J., Beck, N., Bölöni, L, L., Maheswaran, M., Reuther, A, I., and Freund, R, F. (2001) 'A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems'. *Journal of Parallel and Distributed Computing, 61(6) 810-837*

Braun, T. D., Siegel, H, J., Beck, N., Boloni, L., Maheswaran, M., Reuther, A., and Yao, B. (1998) 'A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems'. in *Proceedings of the Seventeenth IEEE Symposium on Reliable Distributed System,* 330-335, IEEE

Braun, T, D., Siegel, H, J., Beck, N., Boloni, L, L., Maheswaran, M., Reuther, A, I., Robertson, J, P., Theys, M, D., Yao, B., Hensgen, D., Freud, R, F. (*2001)* A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing 61 (6) 810–837*

Bryant, R.E. (2007) 'Data Intensive Supercomputing: The Case for DISC'. Technical Report: CMU-CS-07-128, Carnegie Mellon University

Bullnheimer, B., Kotsis, G., and Strauss, C. (1997) *Parallelization strategies for the ant system.* (Technical Report POM-9-97). Vienna, Austria: University of Vienna, Institute of Management Science Also available in (1998). R. De Leone, A. Murli, P. Pardalos, and G. Toraldo (Eds.), *High performance algorithms and software in nonlinear optimization (24)* 87–100. (Series: Applied Optimization, Dordrecht: Kluwer

Buyya, R., Mizuno-Matsumoto, Y., Venugopal, S., and Abramson, D. (2005) 'Neuroscience Instrumentation and Distributed Analysis of Brain Activity Data: A Case for Escience on Global Grids'. *Concurrency and Computation: Practice and Experience* 17 (15), 1783-1798

Buyya, R., and Murshed, M. (2002) 'GridSim: a toolkit for the modelling and simulation of distributed resource management and scheduling for Grid computing'. Concurrency and computation: practice and experience 14(13), 1175-1220

Buyya, R., Abramson, D., and Giddy, J. (2000) 'Nimrod/G: Architecture for a Resource Management and Scheduling System in a Global Computational Grid' in *Proceedings of the Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region*'. IEEE

Buyya, R. (1999) High Performance Cluster Computing: Architectures and Systems (1) Prentice Hall, Upper Saddle *River, NJ, USA*

Buyya, R., Date, S., Mizuno-Matsumoto, Y., Venugopal, S. and Abramson, D. (2004) 'Neuroscience Instrumentation and Distributed Analysis of Brain Activity Data: A case for eScience on Global Grids'. *Journal of Concurrency and Computation: Practice and Experience*

CACI. *Simscript: a simulation language for building large-scale, complex simulation models*, CACI Products Company, San Diego, CA, USA, [online] available from <http://www.simscript.com/simscript.cfm> [9/7/2015]

Caminero, A., Rana, O., Caminero, B., and Carrión, C. (2007) 'An Autonomic Network-Aware Scheduling Architecture for Grid Computing'. in *Proceedings of the 5th International Workshop on Middleware for Grid Computing*. ACM

Caminero, A., Rana, O., Caminero, B., and Carrión, C. (2011) 'Network-Aware Heuristics for Inter-Domain Meta-Scheduling in Grids'. *Journal of Computer and System Sciences* 77 (2), 262-281

Campanini, R., Di Caro, G., Villani, M., D'Antone, I., and Giusti, G. (1994) 'Parallel architectures and intrinsically parallel algorithms: Genetic algorithms'. *International Journal of Modern Physics 5*(1) 95–112

Canabé, M. and Nesmachnow, S. (2012) 'Parallel Implementations of the MinMin Heterogeneous Computing Scheduler in GPU'. *CLEI Electronic Journal* 15 (3), 8-8

Cao, J., Spooner, D, P., Jarvis, S, A., and Nudd, G, R. (2005) 'Grid load balancing using intelligent agents'. *Future generation computer systems*, *21*(1), 135-149

Carretero, J., and Xhafa, F. (2006) 'Use of genetic algorithms for scheduling jobs in large scale Grid applications'. *Technological and Economic Development of Economy*, *12*(1), 11-17

Casanova, H. (2001) 'Simgrid: A Toolkit for the Simulation of Application Scheduling'. in *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Computer Society Press, USA

Casanova, H., and Dongarra, J. (1997) 'NetSolve: A network-enabled server for solving computational science problems.' *International Journal of High Performance Computing Applications 11*(3), 212-223

Casanova, H., Kim, M., Plank, J., Dongarra, J.(1999) 'Adaptive Scheduling for Task Farming with Grid Middleware'. *International Journal of Supercomputer Applications and High Performance Computing*

Casanova, H., Legrand, A., and Quinson, M. (2008) 'SimGrid: A generic framework for Large-Scale Distributed Experiments'. *Computer Modelling and Simulation, UKSIM 2008*, held 1-3 April in Cambridge, UK, 26-131

Catanzaro, B., Fox, A., Keutzer, K., Patterson, D., Su, B., Y., Snir, M., and Chafi, H. (2010) 'Ubiquitous Parallel Computing from Berkeley, Illinois and Stanford'. *Micro,* 30(2), 41-55 *IEEE*

Chaiken, R., Jenkins, B., Larson, P. Å., Ramsey, B., Shakib, D., Weaver, S., and Zhou, J. (2008) 'SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets'. *VLDB Endowment,* 1(2), 1265-1276

Chandra, P., Fisher, A., Kosak, C., Ng, T. E., Steenkiste, P., Takahashi, E., and Zhang, H. (1998) 'Darwin: Customizable resource management for value-added network services'. *in Proceedings of the Sixth International Conference on Network Protocols* 177-188, IEEE

Chapin, S, J., Cirne, W., Feitelson, D, G., Jones, J, P., Leutenegger, S, T., Schwiegelshohn, U., and Talby, D. (1999) 'Benchmarks and standards for the evaluation of parallel job schedulers'. in *Job Scheduling Strategies for Parallel Processing* 67-90, Springer Berlin Heidelberg

Chaudhry, S., Caprioli, P., Yip, S., and Tremblay, M. (2005) 'High-performance throughput computing'. *Micro*, *25*(3), 32-45, *IEEE*

Chavez, A., Moukas, A., and Maes, P. (1997) 'Challenger: A multi-agent system for distributed resources allocation'. in *Proceedings of the first international conference on Autonomous agents*, 323-331 ACM

Chen, J., Li, B., and Wang, E. F. (2014) 'Parallel Scheduling Algorithms Investigation of Support Strict Resource Reservation for Grid '. *Applied Mechanics and Materials* 519, 108-113

Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., and Tuecke, S. (2000) 'The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets'. *Journal of Network and Computer Applications* 23 (3), 187-200

Cho, S., and Jin, L. (2006) 'Managing Distributed, Shared L2 Caches through OS-Level Page Allocation'. in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Micro architecture* 455–468

Ciechanowicz, P., and Kuchen, H. (2010) 'Enhancing Muesli's Data Parallel Skeletons for Multi-Core Computer Architectures'. in *Proceedings of the International Conference on High Performance Computing and Communications (HPCC)*. IEEE

Cirne, W., and Berman, F. (2001) 'A model for moldable supercomputer jobs.' in *Proceedings of the 15th International* Symposium *on Parallel and Distributed Processing* (8), IEEE

Corbalan, J., Martorell, X., and Labarta, J. (2001) 'Improving gang scheduling through job performance analysis and malleability.' in *Proceedings of the 15th international conference on Supercomputing* 303-311, ACM

Colorni, A., Dorigo, M., Maniezzo, V., and Trubian, M. (1994) 'Ant System for job-shop scheduling.' *Belgian Journal of Operations Research, Statistics and Computer Science* (*JORBEL*) *34*, 39–53

Cosnard, M., Jeanot, E., and Yang, T. (1999) 'SLC: symbolic scheduling for executing parameterized task graphs on multimachines'. in *proceedings of the 28th International Conference on Parallel processing*. held in Fukushima, Japan

Creel, M., and Zubair, M. (2012) 'High Performance Implementation of an Econometrics and Financial Application on GPUs'. in *High Performance Computing, Networking, Storage and Analysis* 1147 - 1153 IEEE

Crespo, A., and Garcia-Molina, H. (2002) 'Routing Indices for Peer-to-Peer Systems' . in *Proceedings of the International Conference on Distributed Computing Systems* IEEE

Cybenko, G. (1989) 'Dynamic load balancing for distributed memory multiprocessors'. *Journal of Parallel and Distributed Computing, 7(2), 279-301*

Eck, D, J. (2012) *Introduction to Programming using Java* [online] available from <http://math.hws.edu/javanotes/c12/sl.html> [19 March 2013]

Dean, J., and Ghemawat, S. (2008) 'MapReduce: Simplified Data Processing on Large Clusters'. *Communications of the ACM* 51 (1), 107-113

Dekel, E., and Sahni, S. (1981) 'Binary trees and parallel scheduling algorithms'. 480-492, Springer Berlin Heidelberg

Dekel, E., and Sahni, S. (1983) 'Parallel scheduling algorithms'. *Operations Research* 31(1), 24-49

Dolbeau, R., Bihan, S., and Bodin, F. (2007) 'HMPP: A Hybrid Multi-Core Parallel Programming Environment.' in *Proceedings of the Workshop on General Purpose Processing on Graphics Processing Units (GPGPU)*

Dong, F., and Akl, S, G. (2006) 'Scheduling Algorithms for Grid Computing: State of the Art and Open Problems' *Technical Report No. 2006-504*

Dooley, R., Vaughn, M., Stanzione, D., Terry, S., and Skidmore, E. (2012) 'Software-as-a-Service: The iPlant Foundation API'. in *Proceedings of the 5th IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS)*

Dorigo, M., and Maniezzo, V. (1993) 'Parallel genetic algorithms: Introduction and overview of the current research'. In J. Stender (Ed.) *Parallel genetic algorithms: Theory & applications* 5–42 Amsterdam: IOS Press

Dorigo, M., Maniezzo, V., and Colorni, A. (1996) 'The Ant System: Optimization by a colony of cooperating agents.' *IEEE Transactions on Systems, Man, and Cybernetics—Part B, 26* (1) 29–41

Dorigo, M., Caro, G, D., and Gambardella, L, M. (1999) 'Ant algorithms for discrete optimization.' *Artificial life 5*(2), 137-172

Dorronsoro, B., Bouvry, P., Cañero, J, A., Maciejewski, A, A., and Siegel, H, J. (2010) 'Multiobjective robust static mapping of independent tasks on grids.' *in Proceedings of the IEEE Congress on Evolutionary Computation* (CEC), part of World Conference in Computational Intelligence 3389–3396

Dos Santos, L, P, P. (1996) 'Load distribution: a survey.' *Departamento de Informatica. Universidade do Minho, Portugal*

Du, W., Mummoorthy, M., and Jing, J. (2010) *Uncheatable Grid Computing: Algorithms and Theory of Computation Handbook.* Chapman & Hall/CRC

Eckstein, J. (1994) 'Parallel Branch and Bound Algorithms for General Mixed Integer Programming on the CM-5'. *Technical Report TMC-257, SIAM J. Optim*

Ernemann, C., Hamscher, V., Schwiegelshohn, U., Yahyapour, R., and Streit, A. (2002) 'On Advantages of Grid Computing for Parallel Job Scheduling*'.* in *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE

Etminani, K., and Naghibzadeh, M. (2007) 'A MinMin Max-Min Selective Algorithm for Grid Task Scheduling'. in *Proceedings of the 3rd IEEE/IFIP International Conference on Internet (ICI)'.* IEEE

Feitelson, D, G., and Rudolph, L. (1998) 'Metrics and benchmarking for parallel job scheduling'. in *Job Scheduling Strategies for Parallel Processing* 1-24 Springer Berlin Heidelberg

Feitelson, D., Rudolph, L., and Schwiegelshohn, U. (2004) 'Parallel Job Scheduling – A Status Report'. in *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing.*

Feitelson, D. (2005) *Parallel Workloads Archive* [online] available from <http://www.cs.huji.ac.il/labs/parallel/workload/> [18 March 2015]

Foster, I. (1995) *Parallelism and Computing* [online] available from <http://www.mcs.anl.gov/~itf/dbpp/text/node7.html> [15 October 2011]

Foster, I., and Kesselman, C. (1997) 'Globus: A metacomputing infrastructure toolkit'. *International Journal of High Performance Computing Applications* 11(2), 115-128

Foster, I., and Kesselman, C. (1999) 'The Grid: Blueprint for a New Computing Infrastructure*'.* Morgan Kaufmann, San Francisco', 159-180

Foster, I. (2000) *Internet Computing and the Emerging Grid. Nature Web Matters 7* [online] available from <http://www.nature.com/nature/webmatters/grid/grid.html> [20 September 2011]

Foster, I., Kesselman, C., and Tuecke, S. (2001) 'The anatomy of the grid: Enabling scalable virtual organizations'. *International journal of high performance computing applications 15*(3), 200-222

Foster, I., Kesselman, C., Nick, J. M., and Tuecke, S. (2002) 'Grid Services for Distributed System Integration'. *Computer* 35 (6), 37-46

Foster, I., Kishimoto, H., Savva, A., Berry, D., Djaoui, A., Grimshaw, A., Horn, B., Maciel, F., Siebenlist, F., and Subramaniam, R. (2005) *The Open Grid Services Architecture, Version 1.0* [online] available from<http://www.Ggf.org/documents/Drafts/draft-Ggf-Ogsa-Spec.Pdf> [05/08/2014]

Foster, I., Zhao, Y., Raicu, I., and Lu, S. (2008) 'Cloud Computing and Grid Computing 360-Degree Compared'. in *Proceedings of the Workshop on Grid Computing Environments (GCE2008)* IEEE

Frachtenberg, E., Petrini, F., Coll, S., and Feng, W, C. (2001) 'Gang scheduling with lightweight user-level communication'. in *Parallel Processing Workshops* 339-345, IEEE

Fox, G. (2002) 'Message Passing: From Parallel Computing to the Grid'. *Computers in Science and Engineering*, 4 (5)

Franke, C., Lepping, J., and Schwiegelshohn, U. (2007) 'On advantages of scheduling using genetic fuzzy systems'. In *Job Scheduling Strategies for Parallel Processing* 68-93, Springer Berlin Heidelberg

Freund, R., Taylor, K., Hensgen, D., and Moore, L. (1996) 'Smartnet: A Scheduling Framework for Heterogenous Computing'. in *Proceedings of the Second International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN-96),* IEEE

Freund, R.F., Gherrity, M., Ambrosius, S., Campbell, M., Halderman, M., Hensgen, D., Keith, E., Kidd, T., Kussow, M., Lima, J.D. and Mirabile, F. (1998) 'Scheduling Resources in Multi-User Heterogeneous, Computing Environment with Smartnet'.in *Proceedings of the 7th IEEE Workshop on Heterogeneous Computing* 184-199, IEEE

Frey, J., Tannenbaum, T., Livny, M., Foster, I., & Tuecke, S. (2002) 'Condor-G: A computation management agent for multi-institutional grids'. *Cluster Computing*, *5*(3), 237-246

Fujimoto, N., and Hagihara, K. (2004) 'A comparison among grid scheduling algorithms for independent coarse-grained tasks.' In *International Symposium on Applications and the Internet Workshops* 674-680, IEEE

Geddes, N. (2012) 'The Large Hadron Collider and Grid Computing', *Philosophical Transactions, Series A, Mathematical, Physical, and Engineering Sciences* 370 (1961), 965-977

Geer, D. (2005) 'Chip Makers Turn to Multicore Processors'. *Computer* 38 (5), 11-13

Gepner, P., and Kowalik, M. F. (2006) 'Multi-Core Processors: New Way to Achieve High System Performance'. *Proceedings of theInternational Symposium* on*Parallel Computing in Electrical Engineering (PAR ELEC 2006)* 9-13 IEEE

Gupta, A., Tucker, A., and Urushibara, S. (1991) 'The Impact of Operating System Scheduling Policies and Synchronization Methods of Performance of Parallel Applications'. in *ACM SIGMETRICS Performance Evaluation Review* 19(1), 120-132

Gurudutt, K, V, J. (2013) *Considerations in Software Design for multicore/multiprocessor Architectures. IBM Software Developer* . [online] available from <http://www.ibm.com/developerworks/aix/library/au-aix-multicore-multiprocessor> [06/03/2014]

Hao, Y., Liu, G., Wen, N. (2012) 'An enhanced load balancing mechanism based on deadline control on GridSim'. *Future Generation Computer Systems* 28, 657–665

He, X., Xian-He, S., and Laszewski, G., Von. (2003) 'QoS Guided MinMin Heuristic for Grid Task Scheduling'. *Journal of Computer Science and Technology* 18, 442-451

He, Y., Hsu, W, J., and Leiserson, C, E. (2007) Provably efficient two-level adaptive scheduling. in *Job scheduling Strategies for Parallel Processing* 1-32, Springer Berlin Heidelberg

Hephzibah, D, D, M., and Easwarakumar K, S. (2010) 'A Double Min Min Algorithm for Task metascheduler on Hypercubic P2P Grid Systems'. *International Journal of Computer Science* 7 (4)

Hill, M. D., and Marty, M. R. (2008) 'Amdahl's Law in the Multicore Era'. *Computer* 41 (7), 33-38

Hobbs, L, C., and Theis, D, J. (eds.) (1970) 'A Survey of Parallel Processor Approaches and Techniques, Parallel Processor Systems, Technologies and Application'. New York: Spartan books

Hollander, G. L. (ed.) (1967) 'Architecture for Large Computing Systems'. in *Proceedings of the AFIPS Spring Joint Computer Conference*.ACM,463-466

Hoschek, W., Jaen-Martinez, J., Samar, A., Stockinger, H., and Stockinger, K. (2000) 'Data managemeent in an international data grid project'. In *Grid Computing—GRID 2000* 77-90, Springer Berlin Heidelberg

Hotovy, S. (1996) Workload evolution on the Cornell theory center IBM SP2. In *Job Scheduling Strategies for Parallel Processing* 27-40. Springer Berlin Heidelberg

Howell, F., and McNab R. (1998) 'SimJava: A discrete event simulation package for java with application in computer systems modeling'. in *Proceedings of first international conference on Web-based modeling and simulation*. San Diego CA. Society for computer simulation

Huang, R., Casanova, H., and Chien, A. (2006) 'Using Virtual Grids to Simplify Application Scheduling'. *Proceedings of the 20th International Conference on Parallel and Distributed Processing. held in Rhodes Island, Greece*

Hwang, K., Dongarra, J., and Fox, G. C. (2013) 'Distributed and Cloud Computing: From Parallel Processing to the Internet of Things'. *Morgan Kaufmann*

Kirk, D. B., and Wen-Mei, W. H. (2012) *Programming Massively Parallel Processors: A Hands-on Approach.* Newnes

Ibarra, O H., Kim. C. E., (1977) 'Heuristic Algorithms for Scheduling Independent Tasks on Non-Identical Processors'. *Journal of the Association for Computing Machinery* 24 (2), 280-289

Iosup, A., Li, H., Jan, M., Anoep, S., Dumitrescu, C., Wolters, L., and Epema, D. H. (2008) 'The Grid Workloads Archive'. *Future Generation Computer Systems* 24 (7), 672-686

Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. (2007) 'Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks'. in *ACM SIGOPS Operating Systems Review* 41(3), 59-72

Jada, J. (1992) *An introduction to parallel algorithms*. Addison Wesley

Jaleel, A., Hasenplaugh, W., Qureshi, M., Sebot, J., Steely, S., and Emer, J. (2008) 'Adaptive insertion policies for managing shared caches'. *in Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 208–219.

James, L. (2009) 'Spending Moore's Dividends - Microsoft Research'. *Communications of the ACM'*, 52 (5), 62-69

Jeng, A, A, K., and Lin, B, M. (2005) 'Minimizing the total completion time in single-machine scheduling with step-deteriorating jobs'. *Computers and operations research 32*(3), 521-536

Jeong, C., Choi, Y., Chun, H., Song, S., Jung, H., Lee, S., and Choi, S. (2014) 'Grid-Based Framework for High-Performance Processing of Scientific Knowledge'. *Multimedia Tools and Applications* 71 (2), 783-798

Jin, H., Jespersen, D., Mehrotra, P., Biswas, R., Huang, L., and Chapman, B. (2011) 'High Performance Computing using MPI and OpenMP on Multi-Core Parallel Systems'. *Parallel Computing* 37 (9), 562-575

Jung, G., Gnanasambandam, N., and Mukherjee, T. (2012) 'Synchronous parallel processing of big-data analytics services to optimize performance in federated clouds'. in *Proceedings of the International Conference on Cloud Computing (CLOUD)* 811-818, IEEE

Kalantari, M., and Akbari, M, K. (2009) 'A Parallel Solution for Scheduling of Real Time Applications on Grid Environments'. *Future Generation Computer Systems* 25 (7), 704-716

Kalla, R., Sinharoy, B., and Tendler, J, M. (2004) 'IBM Power5 Chip: A Dual-Core Multithreaded Processor '*. Micro* 24 (2), 40-47

Karatza, H.D. (ed.) (1999) 'Gang Scheduling in a Distributed System with Processor Failures'. *in Proceedings of the UK Performance Engineering Workshop.* held July 22-23 at University of Bristol, UK

Karatza, H. D. (2001). 'Performance analysis of gang scheduling in a distributed system under processor failure'. *International Journal of Simulation: Systems, Science & Technology, UK Simulation Society 2*(1), 14-23.

Karonis, N. T., Toonen, B., and Foster, I. (2003) 'MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface'. *Journal of Parallel and Distributed Computing* 63 (5), 551-563

Karp, A. H. (1987) 'Programming for Parallelism'. *Computer*, 20 (5)

Keat, N, W., Fong, A, T., Chaw, L, T., and Sun, L, C. (2006) 'Scheduling framework for bandwidth-aware job grouping-based scheduling in grid computing'. *Malaysian Journal of Computer Science 19*(2), 117-126

Kenneth, E, K. (1966) 'Changes in Computer Performance'. *Datamation* 12 (9), 40-54

Kessler, C., Dastgeer, U., and Li, L. (2014) 'Optimized Composition: Generating Efficient Code for Heterogeneous Systems from Multi-Variant Components, Skeletons and Containers'. *arXiv preprint arXiv 1405.2915*

Khaled, A., Syed, S., and Kassem, S. (1997) 'Distributed gang scheduling in networks of heterogeneous workstations'. *Computer Communications* 20, 338-348

Khan, K, H., Kalim, Q., and Mostafa, A, E, B. (2014) 'An efficient Grid scheduling strategy for data parallel applications'. *The Journal of Supercomputing* 68 (3), 1487-1502

Kim, S., Chandra, D., and Solihin, Y. (2004) 'Fair cache sharing and partitioning in a chip multiprocessor architecture'. *in proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques 111-122*

Kindervater, G, A, P., and Lenstra, J, K. (1985) 'Parallel Algorithms'. In *Combinatorial Optimization: Annotated Bibliographies,* O'hEigeartaigh, M. ., Lenstra J, K., and Roonoy, K., (eds .) John Wiley, New York, 16-12

Klusacek, D. (2008) *Scheduling in Grid Environment.* Unpublished PhD thesis. Brno: Masary University

Klusáček, D., Rudová, H., Baraglia, R., Pasquali, M., and Capannini, G. (2008) 'Comparison of multi-criteria scheduling techniques'. in *Grid Computing.* Springer, 173-184

Knight, W. (2005) 'Two Heads are Better than One [Dual-Core Processors]'. *IEE Review* 51 (9), 32-35

Kon, F., Campbell, R, H., Mickunas, M., Nahrstedt, K., and Ballesteros, F, J. (2000a) '2K: A distributed operating system for dynamic heterogeneous environments'. in *Proceedings of the Ninth International Symposium on High-Performance Distributed Computing* 201-208, IEEE

Kon, F., Roman, M., Liu, P., Mao, J., Yamane, T., Magalhaes, L., and Campbell, R, H. (2000b) 'Monitoring Security, and Dynamic Configuration with the dynamic TAO Reflective ORB'. *in Proceeedings of the International Conference on Distributed Systems Platforms and Open Distributed Processing (*1795), 121-143, Springer- Verlag

Kondo, M., Sasaki, H., and Nakamura, H. (2007) 'Improving fairness, throughput and energy-efficiency on a chip multiprocessor through DVFS'. *SIGARCH Comput. Archit. News 35(*1), 31–38.

Koziolek, H., Becker, S., Happe, J., Tuma, P., and de Gooijer, T. (2014) 'Towards Software Performance Engineering for Multicore and Manycore Systems'. *ACM SIGMETRICS Performance Evaluation Review* 41 (3), 2-11

Kwiatkowski, J., and Iwaszyn, R. (2010) 'Automatic program parallelization for multicore processors'. *in Parallel Processing and Applied Mathematics* 236-245, Springer Berlin Heidelberg

Kwok, Y, K., and Ahmad, I. (1999) *'*Benchmarking and comparison of the task graph scheduling algorithms*'. Journal of Parallel and Distributed Computing*, 59(3) 381-422

Lawson, B. G., and Smirni, E. (2002) 'Multiple-queue backfilling scheduling with priorities and reservations for parallel systems'. in *Job Scheduling Strategies for Parallel Processing* 72-87, Springer

LeBlanc, R., and Wrinn, M. (2010) 'Adapting Computing Curricula to a Multicore World'. in P*roceedings of IEEE Fronti. Educ. Conference*

Lee, V. W., Kim, C., Chugani, J., Deisher, M., Kim, D., Nguyen, A, D., and Dubey, P. (2010) 'Debunking the 100X GPU myth: an evaluation of throughput computing on CPU and GPU'. in *ACM SIGARCH Computer Architecture News* 38(3), 451-460, ACM

Legrand, A., Marchal, L., and Superieuredelyon, E. (2003) 'Scheduling distributed applications: the Simgrid Simulation model'. *The third IEEE International Symposium on Cluster Computing and the Grid 138-145*

Liang, F., Liu, Y., Liu, H., Ma, S., and Schnor, B. (2013) 'A Parallel Job Execution Time Estimation Approach Based on User Submission Pattern within Computational Grids'. *International Journal of Parallel Programming* 1-5

Lifka, A, D. (1995) 'The ANL/IBM SP Scheduling System'. in *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing (IPPS'95)*. Springer-Verlag

Lin, J., Lu, Q., Ding, X., Zhang, Z., and Zhang, X. (2008) 'Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems'. In *14th International Symposium on High Performance Computer Architecture* 367-378, IEEE

Lin, J., Chen, Y., Jaleel, A., and Tang, Z. (2009) 'Understanding the Memory Behavior of Emerging Multi-Core Workloads'. in Proceedings *of the 8th International Symposium on Parallel and Distributed Computing* IEEE

Lin, J. (2011) 'Scheduling Parallel Tasks with Intra-Communication Overhead in a Grid Computing Environment'. *International Journal of Innovative Computing, Information and Control* 7 (2), 881-896

Litzkow, M., Livny, M., and Mutka, M. (1988) 'Condor – A Hunter of Idle Workstations'. in *Proceedings of the 8th International Conference on Distributed Computing Systems* 104-111, IEEE

Liu, C., Yang, L., Foster, I., and Angulo, D. (2002) 'Design and evaluation of a resource selection framework for Grid applications'. *In Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing 63-72, IEEE*

Liu, C., and Foster, I. (2004) 'A constraint language approach to matchmaking'. In *Proceedings of the 14th International Workshop on Research Issues on Data Engineering: Web Services for e-Commerce and e-Government Applications 7-14, IEEE*

Liu, L., Cui, Z., Xing, M., Bao, Y., Chen, M., and Wu, C. (2012) 'A software memory partition approach for eliminating bank-level interference in multicore systems'. *in Proceedings of the 21st international conference on Parallel architectures and compilation techniques* 367-376, ACM

Liu, H., Abraham, A., and Hassanien, A, E. (2010) 'Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm'. *Future Generation Computer Systems 26*(8), 1336-1343

Liu, Q., and Liao, Y. (2009) 'Grouping-based fine-grained job scheduling in grid computing'. in *Education Technology and computer Science*, *First International Workshop on* (1), 556-559, IEEE

Livny, M., Basney, J., Raman, R., and Tannenbaum, T. (1997) 'Mechanisms for high throughput computing'. *SPEEDUP journal 11*(1), 36-40

Livny, M., and Raman, R. (1999) 'The Grid: Blueprint for a New Computing Infrastructure'. *High-throughput resource management*. *Morgan Kaufmann*, 311-337

Lo, V., Mache, J., and Windisch, K. (1998) 'A comparative study of real workload traces and synthetic workload models for parallel job scheduling'. In *Job Scheduling Strategies for Parallel Processing* 25-46, Springer Berlin Heidelberg

Majo, Z., and Gross, T.R. (2011) 'Memory management in NUMA multicore systems: trapped between cache contention and interconnect overhead'. In *ACM SIGPLAN Notices* (46)11, 11-20, ACM

Luo, P., Lu, K., and Shi, Z. (2007) 'A revisit of fast greedy heuristics for mapping a class of independent tasks onto heterogeneous computing systems'. *Journal of Parallel and Distributed Computing* 67, 695–714

Maheswaran, M., Ali, S., Siegal, H.J., Hensgen, D. and Freund, R. F. (1999) 'Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems'. *Journal of Parallel and Distributed Computing* 59, 107-131

Marinescu, C, D., and Wang, K, Y. (1995) 'On Gang Scheduling And Demand Paging'. in *Proceedings of the Conference on High Performance Computing* New Delhi

McCool, M, D. (2008) 'Scalable Programming Models for Massively Multicore Processors'. *Proceedings of the IEEE 96(5), 816-831*

Mehrotra, S., Shamjith, K., Pandey, P., Asvija, B., and Sridharan, R. (2013) 'A Mechanism to Improve the Performance of Hybrid MPI-OpenMP Applications in Grid'. in *Proceedings of the Conference on High Performance Extreme Computing (HPEC)*IEEE

Mellor-Crummey, J. (2012) CMP522 Multicore Computing, an Introduction *Department of Computer Science, Rice University* [online] available at <http://www.cs.rice.edu/~johnmc/comp522/lecture-notes/COMP522-2014-Lecture1-Introduction.pdf>[22/10/2014]

Messerschmidt, C. M., and Hinz, O. (2013) 'Explaining the Adoption of Grid Computing: An Integrated Institutional Theory and Organizational Capability Approach'. *The Journal of Strategic Information Systems* 22 (2), 137-156

Meyer, R. (2006) 'Emerging Multi-Core Realities'. *Scientific Computing*

Michiko, K. (2013) *Tweeking Moore's Law: Computers of the POST Silicon Era* [online] available from <http://bigthink.com/video/tweaking-moores-law-computer-of-the-post-silicon-era> [03/19 2013]

Mirjalili, S., Mirjalili, S, M., and Lewis, A. (2014) 'Grey wolf optimizer'. *Advances in engineering Software* (*69),* 46-61

Mirsoleimani, S, A., Karami, A., and Khunjush, F. (2013) 'A Parallel Memetic Algorithm on GPU to Solve the Task Scheduling Problem in Heterogeneous Environments'. in *Proceedings of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference'*. ACM

Mizuno, M., Chen, L., and Wallentine, V. (2003) 'Synchronization in a Thread-Pool Model and its Application in Parallel Computing'. in *Proceedings of the PDPTA,* 1879-1885

Monteyne, M. (2008) 'Rapidmind multi-core development platform' *RapidMind Inc., Waterloo, Canada*

Moore, G. (1965) *Cramming More Components onto Integrated Circuits* [online] available from <http://download.intel.com/museum/Moores Law/Articles> [20/09/2011]

Moscicki J. T. (2003) 'Diane-distributed analysis environment for grid-enabled simulation and analysis of physics data'. In *Nuclear Science Symposium Conference Record (3), 1617-1620*

Muralidhara, S. P., Subramanian, L., Mutlu, O., Kandemir, M., and Moscibroda, T. (2011) 'Reducing memory interference in multicore systems via application-aware memory channel partitioning'. *in Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture* 374-385

Muthuvelu, N., Liu, J., Soe, N. L., Venugopal, S., Sulistio, A., and Buyya, R. (2005) 'A dynamic job grouping-based scheduling for deploying applications with fine-grained tasks on global grids'. in *Proceedings of the 2005 Australasian workshop on Grid computing and e-research (44),* 41-48 Australian Computer Society, Inc

Muthuvelu, N., Liu, J., Soe, N. L., Venugopal, S., Sulistio, A., and Buyya, R. (2005) 'A Dynamic Job Grouping-Based Scheduling for Deploying Applications with Fine-Grained Tasks on Global Grids'. in *Proceedings of the 2005 Australasian Workshop on Grid Computing and e-Research* 44

Mutlu, O., and Moscibroda, T. (2008) 'Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems'. *in Proceedings of the 35th Annual International Symposium on Computer Architecture* 63–74

Nakada, H., Sato, M., and Sekiguchi, S. (1999) 'Design and implementations of ninf: towards a global computing infrastructure'. *Future Generation Computer Systems 15(5), 649-658*

Nakada, H., Tanaka, Y., Matsuoka, S., and Sekiguchi, S. (2003) 'Ninf-G: A GridRPC System on the Globus Toolkit'. *Grid Computing: Making the Global Infrastructure a Reality*, 625-637

Neary, M, O., Phipps, A., Richman, S., and Cappello, P. (2000) 'Javelin 2.0: Java-based parallel computing on the Internet'. In *Euro-Par 2000 Parallel Processing* 1231-1238, Springer Berlin Heidelberg

Nesbit, K, J., Laudon, J., and Smith, J, E. (2007) 'Virtual private caches'. *in Proceedings of the 34th annual international symposium on Computer architecture* 57–68

Nesmachnow, S., Cancela, H., and Alba, E. (2012) 'A Parallel Micro Evolutionary Algorithm for Heterogeneous Computing and Grid Scheduling'. *Applied Soft Computing* 12 (2), 626-639

Nesmachnow, S., and Canabé, M. (2011) 'GPU Implementations of Scheduling Heuristics for Heterogeneous Computing Environments'. in Proceedings *of the XVII Congreso Argentino De Ciencias De La Computación.*

Nickolls, J., Buck, I., Garland, M., and Skadron, K. (2008) 'Scalable Parallel Programming with CUDA'. *Queue* 6 (2), 40-53

Oasis Group at INRIA Sohpia-Antipolis (2002) "*Proactive, the java library for parallel, distributed concurrent computing with security and mobility* [online] available from <http://proactive.objectweb.org> [13 May 2015]

Olivier, S, L., Allan, K., Porterfield, K., Wheeler, B., and Jan, F, P. (2011) 'Scheduling task parallelism on multi-socket multicore systems.' *in Proceedings of the 1st International Workshop on Runtime and Operating Systems for Supercomputers* 49-56

Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A. (2008) 'Pig latin: a not-so-foreign language for data processing'. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* 1099-1110, ACM

Osman, I., and Potts, C. (1989) 'Simulated Annealing for Permutation Flow-Shop Scheduling'. *Omega* 17 (6), 551-557

Ousterhout, J. K. (1982) 'Scheduling Techniques for Concurrent Systems'. *in Proceedings of the 3rd International Conference on Distributed Computing Systems,* IEEE

Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A, E., and Purcell, T, J. (2007) 'A survey of general-purpose computation on graphics hardware'. In *Computer graphics forum* 26 (1), 80-113, Blackwell Publishing Ltd

Page, E. H., Litwin, L., McMahon, M, T., Wickham, B., Shadid, M., and Chang, E. (2012) 'Goal-Directed Grid-Enabled Computing for Legacy Simulations'. in *Proceedings of 12th IEEE/ACM International Symposium on*Cluster, Cloud and Grid Computing (CCGrid),IEEE

Papazachos, Z, C., and Karatza, H, D. (2009) 'Scheduling Gangs with Different Distributions in Gangs: Degree of Parallelism in a Multi-Site System'. *in Proceedings of the Fourth Balkan Conference on Informatics. BCI'09,* IEEE

Pardalos, P., and Li, X. (1990) 'Parallel Branch and Bound Algorithms for Combinatorial Optimization'. *Supercomputer* 7(5), 23-30

Parsa, S., and Entezari-Maleki, R. (2009) 'RASA: A New Task Scheduling Algorithm in Grid Environment'. *World Applied Sciences Journal 7, 152-160*

Peng, L., Peir, J, K., Prakash, T., Chen, Y, K., and Koppelman, D. (2007) 'Memory Performance and Scalability of Intel's and AMD's Dual-Core Processors: A Case Study'. in *Proceedings of the 26th IEEE International Performance Computing and Communications Conference (IPCCC),* IEEE

Peng, S., and Nie, Z. (2008) 'Acceleration of the Method of Moments Calculations by using Graphics Processing Unit*s'*. in *Proceedings from the IEEE Transactions on Antennas and Propagation* 56 (7), 2130-2133

Pike, R., Dorward, S., Griesemer, R., and Quinlan, S. (2005) 'Interpreting the data: Parallel analysis with Sawzall'. *Scientific Programming 13*(4), 277-298

Pinel, F., Dorronsoro, B., and Bouvry, P. (2013) 'Solving very Large Instances of the Scheduling of Independent Tasks Problem on the GPU'*. Journal of Parallel and Distributed Computing* 73 (1), 101-110

Ponce, R., Cárdenas-Montes, M., Rodriguez-Vazquez, J, J., Sánchez, E., and Sevilla, I. (2012) *Application of GPUs for the Calculation of Two Point Correlation Functions in Cosmology* [online] available from<http://arxiv.org/pdf/1204.6630.pdf>[22/10/2014]

Prajapati, H, B., and Shah, V, A. (2014) 'Scheduling in Grid Computing Environment'. in *Proceedings of the 4th International Conference on Advanced Computing & Communication Technologies (ACCT 2014*. IEEE

Prasanna, G., Agarwal, A., and Musicus, B, R. (1994) 'Hierarchical compilation of macro dataflow graphs for multiprocessors with local memory'. Trasanction on Parallel and Distributed Systems 5(7), 720-736 IEEE

Qin, X., and Jiang, H. (2005) 'A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs execution on heterogeneous clusters. *Journal of Parallel and Distributed Computing 65(8), 885-900*

Quezada-Pina, A., Tchernykh, A., Gonzalez-Garcia, J,L., Hirales-Carbajal, A., Ramirez-Alcaraz, J, M., Scwiegelshohn, U., and Miranda-Lopez, V. (2012) 'Adaptive Parallel Job Scheduling with Resource Admissible Allocation on Two-Level Hierarchical Grid '. *Future Generation Computer Systems* 28 (7), 965-976

Qureshi, M, K., Lynch, D, N., Mutlu, O., and Patt, Y, N. (2006a) 'A case for MLP-aware cache replacement'. in *Proceedings of the 33rd annual international symposium on Computer Architecture*. IEEE Computer Society, Washington, DC, USA, 167–178

Qureshi, M, K., and Patt, Y, N. (2006b) 'Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared cache. *in proceedings of the 39th Annual IEEE/ACM nterntional Symposium on microarchitecture*

Qureshi, K., Rehman A., and Manuel P. (2011) 'Enhanced GridSim architecture with load balancing'. *Journal of Supercomput 57, 265–275*

Radulescu, A., Nicolescu, C., van-Gemund, A, J., and Jonker, P, P. (2001) 'CPR: Mixed task and data parallel scheduling for distributed systems'. in *Proceedings of the 15th International Symposium on Parallel and Distributed Processing* 9, IEEE

Raman, R., Livny, M., and Solomon, M. (1998) 'Matchmaking: Distributed resource management for high throughput computing'. *in Proceedings of the Seventh International Symposium on High Performance Distributed Computing* 140-146, IEEE

Raman, R., Livny, M., and Solomon, M. (2003) Policy driven heterogeneous resource co-allocation with gangmatchcing. *In Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing* 80-89, IEEE

Ramaswamy, S., Sapatnekar, S., and Banerjee, P. (1997) 'A framework for exploting task and data parallelism on distributed memory multicomputters'. Transaction on Parallel and Distributed Systems 8(11), 1098-1115 IEEE

Ranaweera, S., and Agrawal, D, P. (2001) 'Scheduling of periodic time critical applications for pipelined execution on heterogeneous systems'. *in Proceedings of the International Conference on Parallel Processing131-138*

Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G., and Kozyrakis, C. (2007) 'Evaluating Mapreduce for Multi-Core and Multiprocessor Systems'. in *Proceedings of the 13th International Symposium on High Performance Computer Architecture (HPCA 2007)* IEEE

Ranka, S., Won, Y., and Sahni, S. (1988) *'Programming a hypercube multicomputer', software, (5)5, 69, IEEE*

Ras, B., Chris, J., and Leo, M. (2007) *'Why Browsers Will be Parallel',* Berkeley Parallel Browser Project [online] available from <Http://parallelbrowser.blogspot.com/2007/09/hello-world.html> [15 March 2012]

Rauber, T., and Runger, G. (1998) ‘Compiler support for task scheduling in hierarchical execution models'. *Journal of Systems Architecture* (45), 483-503

Ravi, V.T., Becchi, M., Agrawal, G. and Chakradhar, S. (2012) 'ValuePack: Value-Based Scheduling Framework for CPU-GPU Clusters'. in *Proceedings of the International Conference on High Performance Computing,* Network, Storage and Analysis 53, IEEE

Ridge, E., Kudenko, D., Kazakov, D., and Curry, E. (2005) 'Moving Nature-Inspired Algorithms to Parallel, Asynchronous and Decentralised Environments.' *Self-Organization and Autonomic Informatics 1 (1)*, 35

Ridge, E., Kudenko, D., Kazakov, D., and Curry, E. (2006) 'Parallel, asynchronous and decentralised ant colony system.' in *Proceedings of the First International Symposium on Nature-Inspired Systems for Parallel, Asynchronous and Decentralised Environments (NISPADE)*

Ritchie, G., and Levine, J. (2004) 'A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments'. in *23rd Workshop of the UK Planning and Scheduling Special Interest Group* (PLANSIG 2004)

Rixner, S., Dally, W, J., Kapasi, U, J., Mattson, P., and Owens, J, D. (2000) 'Memory access scheduling'. in *Proceedings of the 27th annual international symposium on Computer architecture* 128–138

Robert, C. (2012) *Could Grid Computing Restore the Internet Growth Curve?* [online] available from < http://www.econstrat.org/publications/op-eds/75-212003-cohen-could-grid-computing-restore-the-internet-growth-curve> [2 December 2012]

Roucairol, C. (1989) 'Parallel Branch and Bound Algorithms-An Overview'. In *Parallel and Distributed Algorithms*. Cosnard, M., Robert, Y., Quinton, P., and Raynal, (eds .) Elsevier Science Publishers, 153-163

Roussopoulos, M., and Baker, M. (2006) 'Practical load balancing for content requests in peer-to-peer networks'. *Journal of Distributed Computing 18*(6), 421-434

Roy, A., and Livny, M. (2004) 'Condor and preemptive resume scheduling'. in *Grid resource management* 135-144, Springer US

Roy, S., De Sarkar, A., and Mukherjee, N. (2014) 'An Agent Based E-Learning Framework for Grid Environment'. in *E-Learning Paradigms and Applications* 121-144, *Springer*

Ryoo, S., Rodrigues, C, I., Baghsorkhi, S, S., Stone, S, S., Kirk, D, B., and Hwu, W, M, W. (2008) 'Optimization Principles and Application Performance Evaluation of a Multithreaded GPU using CUDA*'.* in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming 73-82,* ACM

Sabin, G., Kettimuthu, R., Rajan, A., and Sadayappan, P. (2003) 'Scheduling of Parallel Jobs in a Heterogeneous Multi-Site Environment'. in *Job Scheduling Strategies for Parallel Processing* 87-104, Springer

Sajedi, H., and Rabiee, M. (2014) 'A Metaheuristic Algorithm for Job Scheduling in Grid Computing'. *International Journal of Modern Education and Computer Science (IJMECS)* 6 (5), 52

Segal, B., Robertson, L., Gagliardi, F., Carminati, F. and Cern, G. (2000) 'Grid computing: The European data grid project'. In *IEEE Nuclear Science Symposium and Medical Imaging Conference* 1(2)

Selvi, S, T., Kumari, M., Prabavathi, K., and Kannan, G. (2010) 'Estimating job execution time and handling missing job requirements using rough set in grid scheduling'. In *International Conference on Computer Design and Applications (ICCDA)* 4(4), 295, IEEE

Schauer, B. (2008) 'Multicore processors–a Necessity'. *ProQuest Discovery Guides* 1-14

Schmidt, H, A., Strimmer, K., Vingron, M., and von Haeseler, A. (2002) 'TREE-PUZZLE: maximum likelihood phylogenetic analysis using quartets and parallel computing'. *Bioinformatics* 18(3), 502-504

Schmidt, D, C., and Cleeland, C. (1999) 'Applying Patterns to Develop Extensible ORB Middleware'. *Communications Magazine Special Issue on Design Patterns* 37(4), 54–63

Schwiegelshohn, U., Badia, R, M., Bubak, M., Danelutto, M., Dustdar, S., Gagliardi, F., Geiger, A., Hluchy, L., Kranzlmueller, D., Laure, E., Priol, T., Reinefeld, A., Reuter, A., Rienhoff, O., Rueter, T., Sloot, P., Talia, D., Ullmann, K., Yahyapour, R., and von Voigt, G. (2010) 'Perspectives on Grid Computing'. *Future Generation Computer Systems* (26)8, 1104-1115

Shah, S, N, M., Mahmood, A, K, B., and Oxley, A. (2011) 'Dynamic Multilevel Hybrid Scheduling Algorithms for Grid Computing'. *Procedia Computer Science* 4, 402-411

Shah, S, N, M., Mahmood, A, K, B., Oxley, A., and Zakaria, M, N. (2012) 'QoS based performance evaluation of grid scheduling algorithms'. in *Proceedings of the International Conference on Computer and Information Science (ICCIS)* 2, 700-705 IEEE

Sharma, R., Soni, V, K., Mishra, M, K., Bhuyan, P., and Dey, U, C. (2010) 'An Agent Based Dynamic Resource Scheduling Model with FCFS-Job Grouping Strategy in Grid Computing'. *Waset, ICCGCS*

Shivaratri, N, G., Krueger, P., and Singhal, M. (1992) 'Load distributing for locally distributed systems' *Computer* 12 (25), 33–44

Shu, W., and Wu, M, Y. (1996) 'Runtime incremental parallel scheduling (RIPS) on distributed memory computers'. IEEE Transaction on Parallel and Distributed Systems (6), 637-649

Siegel, H, J., Siegel, L, J., Kemmerer, F, C., Mueller Jr, P, T., Smalley Jr, H, E., and Smith, S, D. (1981) 'PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition'. *IEEE Transactions on Computers*, *100* (12), 934-947

Singh, C., and Agrawal, N. (2014) 'Comparison among Different Task Duplication Scheduling Algorithm in Grid Computing System'. *International Journal of Engineering Trends and Technology (IJETT)* 10(4)

Sodan, A, C., Doshi, C., Barsanti, L., and Taylor, D. (2006) 'Gang scheduling and adaptive resource allocation to mitigate advance reservation impact'. in the *Sixth International Symposium on Cluster Computing and the Grid* 1(5) IEEE

Song, H., Liu, X., Jakobsen, D., Bhagwan, R., Zhang, X., Taura, K., and Chien, A. (2000) 'The MicroGrid:

A scientific Tool for Modeling Computational Grids'. in Proceedings of IEEE Supercomputing 4-10

Soni, V, K., Sharma, R., Mishra, M, K., and Das, S. (2010) 'Constraint-based job and resource scheduling in grid computing'. In *Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT,* (4), 334-337 IEEE

Soni, V. K., Sharma, R., and Mishra, M. K. (2010). 'Grouping-based job scheduling model in grid computing'. *World Academy of Science, Engineering and Technology 41*, 781-784

Stone, J, E., Gohara, D., and Shi, G. (2010) 'OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems'. *Computing in Science and Engineering* 12 (3), 66

Stone, J.E., Phillips, J.C., Freddolino, P.L., Hardy, D.J., Trabuco, L.G. and Schulten, K., 2007. Accelerating molecular modeling applications with graphics processors. *Journal of computational chemistry*, *28*(16), pp.2618-2640.

Stone, J, E., Phillips, J, C., Freddolino, P, L., Hardy, D, J., Trabuco, L, G., and Schulten, K. (2007) 'Accelerating Molecular Modeling Applications with Graphics Processors'. *Journal of computational chemistry 28*(16), 2618-2640

Stutzle, T. (1998) 'Parallelization strategies for ant colony optimization'. *Proceedings of Fifth International Conference on Parallel Problem Solving from Nature* 722–731. Berlin, Springer-Verlag

Stutzle, T., and Hoos, H. (1997a) 'Improvements on the Ant System: Introducing *MAX*¡*MIN* ant system'. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms* 245–249, Springer-Verlag

Stutzle, T., and Hoos, H. (1997b) 'The *MAX* ¡*MIN* Ant System and local search for the traveling salesman problem'. in *Proceedings of IEEE International Conference on Evolutionary Computation and Evolutionary Programming* 309–314

Subhlok, J., and Vondran, G. (2000) 'Optimal use of mixed task and data parallelism for pipelined computations*'. Journal of Parallel and Distributed Computing 60, 297-319*

Suh, G, E., Rudolph, L., and Devadas, S. (2004) 'Dynamic partitioning of shared cache memory'. *Journal of Supercomputing 28(1), 7-26*

Sulistio, A., Cibej, U., Venugopal, S., Robic, B., and Buyya R. (2007) 'A Toolkit for Modelling and Simulating Data Grids: An Extension to GridSim'. *Concurrency and Computation: Practice and Experience* 20(13), 1591 - 1609

Sutter, H. (2005) 'The Free Lunch is Over: 'A Fundamental Turn Toward Concurrency in Software'. *Dr. Dobb's Journal* 30 (3), 202-210

Tam, D. K., Azimi, R., Soares, L. B., and Stumm, M. (2009) 'RapidMRC: approximating L2 miss rate curves on commodity systems for online optimizations'. *in Proceeding of the 14th international conference on Architectural support for programming languages and operating systems* 121–132

Tang, X., Li, K., Qiu, M., and Sha, E, H. (2012) 'A Hierarchical Reliability-Driven Scheduling Algorithm in Grid Systems'. *Journal of Parallel and Distributed Computing* 72 (4), 525-535

Tannenbaum, T., Wright, D., Miller, K., and Livny, M. (2001) 'Condor: a distributed job scheduler'. in *Beowulf cluster computing with Linux* 307-350, MIT press

Tech4globe (2010) *What is Grid Computing? Tech for Globe* [online] available from <http://www.tech4globe.com/what-is-grid-computing.html> [02 March 2015]

Tendulkar, P. (2014) *Mapping and Scheduling on Multi-Core Processors using SMT Solvers.* Unpublished PhD thesis. Grenoble University

Thain, D., Tannenbaum, T., and Livny, M. (2005) 'Distributed computing in practice: The Condor experience.' *Concurrency-Practice and Experience 17*(2-4), 323-356

Thurber, K, J., and Wald, L, D. (1975) 'Associative and Parallel Processors'. *ACM Computing Surveys (CSUR)* 7(4), 215-255

Tchernykh, A., Ramírez, J, M., Avetisyan, A., Kuzjurin, N., Grushin, D., and Zhuk, S. (2006) 'Two level job-scheduling strategies for a computational grid'. In *Parallel Processing and Applied Mathematics* 774-781, Springer Berlin Heidelberg

Trienekens, H, W, J, M., and De Bruin, A. (1992) 'Towards a taxonomy of parallel Branch and Bound'. Report EUR-CS- 92-01, Computer science department, Faculty of Economics, Erasmus University, Rotterdam, Netherland

Tsaregorodtsev, A., Garonne, V., and Stokes-Rees, I. (2004) 'Dirac: A scalable lightweight architecture for high throughput computing. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing* 19-25, IEEE Computer Society

Tuomenoksa, D, L., and Siegel, H, J. (1981) 'Application of two-dimensional bin packing algorithms for task scheduling in the PASM multimicrocomputer system.' In *Nineteenth Allerton Conference on Communication, Control and Computing* (542)

Ungurean, I. (2015) 'Job Scheduling Algorithm based on Dynamic Management of Resources Provided by Grid Computing Systems'. *Elektronika ir Elektrotechnika 103*(7), 57-60

Varga, A. (2001) 'The OMNeT++ Discrete Event Simulation System'. in *Proceedings of the European Simulation Multiconference (ESM 2001)*, held. June 6-9, 2001, Prague, Czech Republic

Vazhkudai, S., Tuecke, S., and Foster, I. (2001) 'Cluster Computing and Grid'. in *Proceedings of the 1st IEEE/ACM International Symposium on Replica Selection in the Globus Data Grid* IEEE

Venugopal, S. and Buyya, R. (2008) 'An SCP-Based Heuristic Approach for Scheduling Distributed Data-Intensive Applications on Global Grids'. *Journal of Parallel and Distributed Computing* 68 (4), 471-487

Viry, P. (2010) 'Ateji PX for Java-Programming made simple'. Ateji White Paper

Viry, P. (2011) 'Parallel and distributed programming extensions for mainstream languages based on pi-calculus'. in Proceedings of the *30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing* 343-344

Wang, P, H., Collins, J, D., Chinya, G, N., Jiang, H., Tian, X., Girkar, M., and Wang, H. (2007) 'EXOCHI: Architecture and Programming Environment for a Heterogeneous Multi-Core Multithreaded System'. *ACM SIGPLAN Notices* 42 (6), 156-166

Widmer, M., and Hertz, A. (1989) 'A New Heuristic Method for the Flow Shop Sequencing Problem'. *European Journal of Operational Research* 41 (2), 186-193

Wieczorek, M., Hoheisel, A., and Prodan, R. (2009) 'Towards a General Model of the Multi-Criteria Workflow Scheduling on the Grid'. *Future Generation Computer Systems* 25 (3), 237-256

Wiseman, Y., and Feitelson, D, G. (2003) 'Paired Gang Scheduling'. *IEEE Transactions on Parallel and Distributed Systems* 4 (6), 581-592

Wu, M, Y., Shu, W., and Chen, Y. (2000) 'Runtime parallel incremental scheduling of DAGS'. *in proceedings of the International Conference on Parallel Processing* 541-548

Wu, M., Shu, W., and Zhang, H. (2000) 'Segmented MinMin: A static mapping algorithm for metatasks on heterogeneous computing systems'. in *Proceedings of the 9th Heterogeneous Computing Workshop* 375, Washington, DC, USA, IEEE Computer Society

Xhafa, F., and Abraham, A. (2010) 'Computational models and heuristic methods for Grid scheduling problems'. *Future generation computer systems 26*(4), 608-621

Xhafa, F., Alba, E., Dorronsoro, B., Duran, B. (2008a) 'Efficient batch job scheduling in grids using cellular memetic algorithms'. *Journal of Mathematical Modelling and Algorithms* 7 (2), 217-236

Xhafa, F., Carretero, J., Alba, E., Dorronsoro, B. (2008b) 'Design and evaluation of tabu-search method for job scheduling in distributed environments'. in *Nature Inspired Distributed Computing (NIDISC) sessions of the International Parallel and Distributed Processing Symposium (IPDPS) 2008 Workshop* 2319–2326 IEEE

Xia, H., Casanova, H., and Chien, A. (1999) 'The MicroGrid: Using online simulation to predict application performance in diverse grid network environment'. *The 2nd International Workshop on challenges of Large Applications in Distributted Environment.* held in Washington, DC, USA

Xiao, P., and Dongbo, L. (2014) 'Multi-Scheme Co-Scheduling Framework for High-Performance Real-Time Applications in Heterogeneous Grids'. *International Journal of Computational Science and Engineering* 9 (1), 55-63

Xie, T., and Qin, X. (2005) 'Enhancing security of real-time applications on grids through dynamic scheduling'. In *Job Scheduling Strategies for Parallel Processing* 219-237, Springer Berlin Heidelberg

Xin, L., Xia, H., and Chien, A. (2004) 'Validating and Scaling the MicroGrid: A Scientific Instrument for Grid Dynamics.' *Journal of Grid Computing 2(2), 141-161*

Yang, H., and Tate, M. (2009) 'Where are we at with Cloud Computing? A Descriptive Literature Review'. in *Proceedings of the 20thAustralasian Conference on Information System2-4*

Ye, G., Rao, R., and Li, M. (2006) 'A multiobjective resources scheduling approach based on genetic algorithms in grid environment'. In *Fifth International Conference on Grid and Cooperative Computing Workshops* 504-509, IEEE

Yu, X., and Yu, X. (eds.) (2009) 'A New Grid Computation-Based MinMin Algorithm.' in Proceedings of the *6th International Conference on Fuzzy Systems and Knowledge Discovery FSKD'09*. IEEE

Yu, Y., Isard, M., Fetterly, D., Budiu, M., Erlingsson, Ú., Gunda, P, K., and Currey, J. (2008) 'DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language. In *OSDI* (8) 1-14

Zhang, Y., Franke, H., Moreira, J, E., and Sivasubramaniam, A. (2000) 'The Impact of Migration on Parallel Job Scheduling for Distributed Systems'. in *Euro-Par 2000 Parallel Processing*, Springer

Zhang, W., and Cheng, A, M. (2006) 'Multisite Co-Allocation Algorithms for Computational Grid'. in *Proceedings of the 20th International Symposium on Parallel and Distributed Processing IPDPS, IEEE*

Zhang, X., Dwarkadas, S., and Shen, K. (2009) 'Towards practical page coloring-based multicore cache management'. *in Proceedings of the 4th ACM European conference on Computer systems* 89–102

Zhou, B, B., Walsh, D., and Brent, R, P. (2000) 'Resource Allocation Schemes for Gang Scheduling'. in J*ob scheduling strategies for Parallel Processing* 1911, 74-86, Springer Berlin Heidelberg

Zhoujun, H., Zhigang, H., and Zhenhua, L. (2010) 'A Service-Clustering-Based Dynamic Scheduling Algorithm for Grid Tasks'. *International Journal of Grid and Distributed Computing* 3 (3), 53-65

Zhuravlev, S., Saez, J.C., Blagodurov, S., Fedorova, A. and Prieto, M., 2012. Survey of scheduling techniques for addressing shared resources in multicore processors. *ACM Computing Surveys (CSUR)*, *45*(1), p.4.

Zhuravlev, S., Saez, J, C., Blagodurov, S., Fedorova, A, A., and Prieto, M. (2012) 'Survey of scheduling techniques for addressing shared resources in multicore processors'. *ACM Computing Surveys* 45(1)

Glossary

Glossary

Blocking refers to situation when a process in execution that requires data (or input) in order to continue waits for the input and continues execution after the input is supplied

Coarse grain granularity refers to a situation where the percentage of computational work done is far greater than the time used for communication

Distributed computing system is a virtual computer formed by a networked set of heterogeneous machines that agree to share their local resources with each other

Embarrassingly parallel: These are parallel systems with the ability to solve many independent tasks simultaneously with no need for any coordination amongst the processors.

Fine grain granularity refers to a situation where the percentage of computational work done is relatively small compared to the time used for communication

High-throughput computing (HTC) is a computing paradigm that delivers processing deadline by employing several data-level parallelisms to process data independently on different processing elements using a similar set of operations

Granularity: Granularity in parallel programming describes the ratio between computation time and communication time

The **Grid** is an aggregation and integration of heterogeneously diverse computing systems, clusters and powerful computers (by a set of protocols) into a virtual unit that combines to provide seamless computing utility services to meet the need of users via a fast transfer mechanism

Grid resources are computing machines or processing elements or memory devices on the Grid which offers computer processing or storage power to consumers

A **job group** is a collection of users' jobs, in the context of this research, it is a collection of users' jobs intended to be scheduled for execution on the Grid

A **machine** or **resources group** contains a set of different computers or Grid resources categorised by the algorithm for servicing a set of jobs from a job group – the machines are grouped based on their configuration.A group of machine or group of Grid resources therefore comprises a list of machines from various Grid sites but having similar or varying configurations depending on the grouping method used

Makespan refers to the combined time taken to schedule and execute a group of job.

Massively parallel: These are computer systems with many processors that are synchronized and coordinated to execute tasks in parallel

Match-making is the means by which Resource Requests and Resource Owners that satisfy each other are identified and paired together.

M-task is a task that can be run on a multiple processor computer.

Multicore systems are computers that are furnished with several execution cores on one CPU, this allows for multiple level of parallelism by the cores. This is referred to as chip-level multiprocessing (CMP)

Multiprocessor systems have several CPUs that allow them to process simultaneously in parallel. This is referred to as simultaneous multiprocessing (SMP).

Multi-scheduling refers to the simultaneous election or selection of several independent jobs from different groups and dispatching to several different Grid resources for execution

Multithreading is an execution model that allows multiple executions of threads such that they execute independently but share their process resources

Non-clairvoyant scheduling this is the scheduling of jobs without prior knowledge of the execution time of the jobs

Non-Parallelizable: This refers to algorithms that can never be parallelized. With such algorithms, parallelization cannot result in any speedup

Parallelism or parallel computing is the ability of computer processors to work cooperatively and simultaneously on a task or on multiple tasks.

Parallelizable algorithm: This is an algorithm that can be made to execute in parallel.

Parallel overhead: This is the amount of time required to coordinate parallel tasks instead of doing useful work. Parallel overheads can be caused by factors like synchronization, data communication, task start up time and task termination time

Process is an executing program or a running program

Scalability: Scalability in parallelism refers to the ability of a system to increase or decrease its performance according to job loads without a detrimental effect on the quality of service. It also refers to the ability of a parallel system to proportionally increase in parallelism speedup with the addition of more resources. This is influenced by factors like hardware, application program, parallel overhead and characteristics of the application

Scheduling is the allocation of limited resources to contending demands from processes based on policies and rules that serve to ensure that certain standards are adhered to. Within a computing system, the limited resources could be processors, memory, input and output media and the contending demands arise from the several processes executing within. On the Grid, the resources (processors and memory) are aggregated from various locations and deemed to be available. The contending requirements are no longer the processors but users' submitted jobs and associated requests. Hence the requirement for scheduling on the Grid becomes how to manage the available resources to meet the contending users' submitted jobs and associated requests rather than how to manage processors between processes

Space-sharing is the actual scheduling of cores to execute the thread chosen to run at the time

S-task is a task that can run only on a single processor computer

Task is a piece of work that needs to be performed

Thread is a light weigh process

Time-sharing is the scheduling of threads to execute on processors at time intervals

Throughput refers to the number of jobs processed or scheduled within a given time

User jobs are the jobs or processes submitted by users onto the Grid for processing

Appendices

Appendix A: Header File from the Grid Workloads Archive

This appendix shows a header file from the Grid Workloads Archive (Anoep et al. 2007). It also shows some sample data from a trace in Grid Workload Format (GWF).

B1. Header File

Generated by get-clean-log.py ($Revision: 0.1$) on Tue February 20, 2007, at 09:48:14 PM
Authors: AlexandruIosup and Mathieu Jan ({A.Iosup|M.Jan} at tudelft.nl)
The Grid Workloads Archive (http://gwa.ewi.tudelft.nl/)
External coallocated_jobs info file: Grid5000_coallocated_jobs.log
External interactive_jobs info file: Grid5000_interactive_jobs.log
External reservation_jobs info file: Grid5000_reservation_jobs.log
External sites_time info file: Grid5000_sites_time.log
External user_to_group info file: Grid5000_user_to_group.log
Format documentation: Grid Workload Format (http://gwa.ewi.tudelft.nl/)
Field description from left to right:

# 1 JobID	counter
# 2 SubmitTime	in seconds, starting from zero
# 3 WaitTime	in seconds
# 4 RunTime	runtime measured in wall clock seconds
# 5 NProcs	number of allocated processors
# 6 AverageCPUTimeUsed	average of CPU time over all allocated processors
# 7 Used Memory	average per processor in kilobytes
# 8 ReqNProcs	requested number of processors
# 9 ReqTime:	requested time measured in wall clock seconds
# 10 ReqMemory	requested memory (average per processor)
# 11 Status	job completed = 1, job failed = 0, job cancelled = 5
# 12 UserID	string identifier for user
# 13 GroupID	string identifier for group user belongs to

# 14 ExecutableID	name of executable
# 15 QueueID	string identifier for queue
# 16 PartitionID	string identifier for partition
# 17 OrigSiteID	string identifier for submission site
# 18 LastRunSiteID	string identifier for execution site
# 19 JobStructure	single job = UNITARY, composite job = BoT
# 20 JobStructureParams	if JobStructure = BoT, contains batch identifier
# 21 UsedNetwork	used network resources in kilobytes/second
# 22 UsedLocalDiskSpace	in megabytes
# 23 UsedResources	list of comma-separated generic resources (ResourceDescription:Consumption)
#	c.q. memory usage in Gb seconds, io data transferred, and io wait time in seconds
# 24 ReqPlatform	CPUArchitecture,OS,OSVersion
# 25 ReqNetwork	in kilobytes/second
# 26 ReqLocalDiskSpace	in megabytes
# 27 ReqResources	list of comma-separated generic resources (ResourceDescription:Consumption)
# 28 VOID	identifier for Virtual Organization
# 29 ProjectID	identifier for project

(fields contain -1 if not available)

B2. Data Sample from a GWF Trace File

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	JobId	SubmitTime	WaitTime	RunTime	NProc	AverageCPUTimeU	UsedMemory	ReqNProc	ReqTime	ReqMem	Status	UserId	GroupId	ExecutableId	QueueId
2															
3	0	1083658801	1	0	4	-1	-1	4	3600	-1	1	user386	group4	app34	queue0
4	1	1083658849	1	19	1	-1	-1	1	3600	-1	1	user112	group6	app0	queue0
5	2	1083658875	2	10	5	-1	-1	5	3600	-1	1	user112	group6	app0	queue0
6	3	1083658891	5	8	90	-1	-1	90	3600	-1	1	user112	group6	app0	queue0
7	4	1083658911	5	19	100	-1	-1	100	3600	-1	1	user112	group6	app0	queue0
8	5	1083658944	1	25	1	-1	-1	1	3600	-1	0	user112	group6	app0	queue0
9	6	1083659210	1	6	1	-1	-1	1	3600	-1	1	user568	group6	app0	queue0
10	7	1083659322	1	43205	4	-1	-1	4	43200	-1	0	user386	group4	app0	queue0
11	8	1083659636	1	5	1	-1	-1	1	3600	-1	1	user568	group6	app0	queue0
12	9	1083660389	-1	-1	4	-1	-1	4	9000	-1	0	user267	group5	app507	queue48
13	10	1083660523	2	156	7	-1	-1	7	18000	-1	1	user569	group6	app0	queue0
14	11	1083660693	1	19	7	-1	-1	7	18000	-1	1	user569	group6	app0	queue0
15	12	1083660719	1	4	7	-1	-1	7	18000	-1	1	user569	group6	app0	queue0
16	13	1083660726	1	2801	7	-1	-1	7	18000	-1	1	user569	group6	app0	queue0
17	14	1083660777	1	1	4	-1	-1	4	3600	-1	1	user267	group5	app507	queue0
18	15	1083660832	1	0	4	-1	-1	4	3600	-1	1	user267	group5	app507	queue0
19	16	1083660933	1	0	4	-1	-1	4	3600	-1	1	user267	group5	app507	queue0
20	17	1083661197	1	20992	1	-1	-1	1	36000	-1	1	user570	group6	app0	queue0
21	18	1083661769	1	23027	1	-1	-1	1	43200	-1	1	user571	group6	app0	queue0
22	19	1083661777	1	23014	1	-1	-1	1	43200	-1	1	user571	group6	app0	queue0
23	20	1083662072	1	22216	1	-1	-1	1	28800	-1	1	user67	group2	app0	queue0
24	21	1083663533	1	13734	10	-1	-1	10	18000	-1	0	user569	group6	app0	queue0
25	22	1083669430	1	1	4	-1	-1	4	3600	-1	1	user267	group5	app507	queue0
26	23	1083669460	1	1	4	-1	-1	4	3600	-1	1	user267	group5	app507	queue0
27	24	1083669628	1	1	4	-1	-1	4	3600	-1	1	user267	group5	app507	queue0
28	25	1083669739	1	1	1	-1	-1	1	3600	-1	1	user267	group5	app507	queue0
29	26	1083669841	1	0	1	-1	-1	1	9000	-1	1	user267	group5	app507	queue0
30	27	1083670043	1	1	4	-1	-1	4	9000	-1	1	user267	group5	app507	queue0
31	28	1083670602	1	1	4	-1	-1	4	9000	-1	1	user267	group5	app507	queue0
32	29	1083670874	1	1	4	-1	-1	4	9000	-1	1	user267	group5	app507	queue0
33	30	1083671023	1	1	4	-1	-1	4	9000	-1	1	user267	group5	app507	queue0
34	31	1083677193	1	2	1	-1	-1	1	3600	-1	1	user569	group6	app508	queue0
35	32	1083677217	1	248	3	-1	-1	3	3600	-1	1	user109	group6	app0	queue0
36	33	1083677226	1	7160	5	-1	-1	5	18000	-1	1	user569	group6	app0	queue0
37	34	1083679821	1	3	4	-1	-1	4	9000	-1	1	user267	group5	app507	queue0
38	35	1083680531	2	49	16	-1	-1	16	3600	-1	1	user572	group6	app509	queue0
39	36	1083681049	1	911	8	-1	-1	8	900	-1	0	user572	group6	app509	queue0
40	37	1083681201	1	3	4	-1	-1	4	9000	-1	1	user267	group5	app507	queue0

P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
Partition	OrigSiteI	LastRunSiteI	JobStruct	JobStru	UsedNetw	UsedLocalD	UsedResc	ReqPlatf	ReqNetwo	Request	Request	Virtual	ProjectId
-1	G1/site4	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site4	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site5	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site5	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site5	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site5	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site2	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site5	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site5	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site5	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site5	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site5	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site5	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site5	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site5	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site5	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site5	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site6	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	G1/site5	G1/site6/c1	UNITARY	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Appendix B: Grid Workloads Archive Acknowledgement

The researcher is grateful to the following groups and members of their teams for making the Grid Workloads Archive available and free to researchers and developers alike:

- The Parallel and Distributed Systems Group at Delft University of Technology (TUDelft), Netherlands (GWA 2014) (http://www.pds.ewi.tudelft.nl/). Members of the group are:ShannyAnoep (TU Delft); CatalinDumitrescu (TU Delft); Dick Epema (TU Delft);AlexandruIosup (TU Delft); Mathieu Jan (TU Delft); Hui Li (U. Leiden); and Lex Wolters (U. Leiden).

- The e-Science Group of HEP at Imperial College London for providing the LCG data and Hui Li for making the data publicly available and DrFeitelson of the parallel workloads archive. http://lcg.web.cern.ch/LCG

- The Grid'5000 team (especially Dr. Franck Cappello) and the OAR team especially Dr. Olivier Richard and Nicolas Capit for the trace http://oar.imag.fr . Also special thanks to John Morton (john_x_sharrcnet.ca) for providing the trace file and for making the parallel workload archive publicly available

- The AuverGrid team with special thanks to Dr. Emmanuel Medernach, the owner of the AuverGrid system made available through the Grid workloads archive http://auvergrid.fr

- NorduGrid team, with special thanks to Dr. BalaszKnoya, the owner of NorduGrid system made public through the Grid workload archive.

Appendix C: Selected Job Scheduling Algorithms on the Grid

This appendix describes some selected scheduling algorithms from the literature review. The algorithms have been selected so as to give representation to each of the categories in the literature review. The categories were:

- Classical Grid Scheduling Algorithms
- Fusion and enhancement of the Classical Algorithm
- QoS Focused Algorithms
- Adaptive Grid Scheduling Algorithms
- Scheduling Algorithms based on Nature

Classical Algorithms		
Algorithm/Characteristics	**Simulation/scenario**	**Performance Result**
MinMin (Ibarra and Kim 1977) This algorithm schedules a set of tasks onto a set of machines in such a way that the task with the smallest completion time on any machine is assigned to that machine. When the task has been assigned the remaining tasks and all machines are looked again and the process repeats. This is why it is called MinMin the smallest task out of the tasks remaining is assigned to the machine that can complete it the fastest. Smaller jobs are thus favoured. The algorithm optimises the finishing time of all the jobs on the processors. If the finishing time of the jobs on all processors are the same, then the schedule is optimal but if any processor is idle while the others are not, then the schedule may not be optimal.	There was no simulation carried out because this was a theoretical study.	A simplified scheduling problem involving identical processors and restricted task sets was shown to be P-complete. A least processing time algorithm (e.g. like MaxMin or MinMin) applied to this problem produces schedules which are near optimal (even load and shortest completion time) for large N (where N is the number of tasks).
Classical Algorithms		
Algorithm/Characteristics	**Simulation/scenario**	**Performance Result**
MaxMin (Ibarra and Kim 1977) The MaxMin differs from the MinMin in that instead of assigning the task with the earliest completion time, it selects the task with the latest or maximum completion time and assigns it to the machine that can process it the	There was no simulation carried out because this was a theoretical study.	A simplified scheduling problem involving identical processors and restricted task sets was shown to be P-complete. A least processing time

fastest. Hence the name MaxMin. Expectedly, these are always the larger tasks. Hence, this algorithm favours larger tasks.		algorithm (e.g. like MaxMin or MinMin) applied to this problem produces schedules which are near optimal (even load and shortest completion time) for large N (where N is the number of tasks).
Sufferage (Maheswaran et al. 1999) Sufferage was a new algorithm for batch mode proposed by the researchers. The Sufferage heuristic is based on the idea that better makespan can be achieved if a machine is assigned to a task that would 'suffer' mostin terms of expected completion time if that particular machine is not assigned to it. The sufferage value of a task is defined as the difference between the second earliest completion time of a task of some machine and the earliest completion time of that task on the same machine.	The researchers compared new and previously proposed dynamic matching and scheduling heuristics for mapping independent tasks onto heterogeneous computing systems under a variety of simulated computational environments. Three new heuristics, one for batch mode and two for immediate mode, were introduced as part of this research. Simulation studies were performed to compare theseheuristics with some existing ones. If the sufferage value of task t_i is the difference between its second earliest completion time on machine m_y and its earliest completion time on another machine $m_{x,}$ $_T$hen using $m_{x,}$ will result in the best completion time for t_i.	The Sufferage algorithm performed better than MinMin and MaxMin but only slightly better than MinMin The simulation revealed that the choice of which dynamic mapping heuristic to use in a given heterogeneous environment depends on the structure of the heterogeneity among tasks and machines.

Fusion and Enhancement Algorithms		
Algorithm/Characteristics	**Simulation/scenario**	**Performance Result**
Multiple-Queue Backfilling Scheduling with Priorities and Reservations for Parallel Systems Lawson and Smirni (2002) This algorithm proposes a non-FCFS policy to schedule parallel job on Grid systems. The algorithm monitors the intensity and variability of the incoming jobs to the Grid and adapts the scheduling parameters according to the variables. The method reduces resource fragmentation by employing backfilling to enable jobs execute before other jobs that arrive earlier than they did and are in front of them on the queue. Resource fragmentation arises when there are idle processors while a job or jobs keeps waiting chiefly because the available processor does not meet their processing requirement. Two known methods of backfilling are aggressive and conservative backfilling. Aggressive backfilling permits jobs to backfill as long as it does not delay the first job in the queue. While conservative backfilling permits a job to back fill only when it is guaranteed that it does not delay any previous job in the queue	The work considered two categories of jobs. First is that jobs submitted by local users are given high priority and jobs submitted by external users (not within the providing Grid) are granted low priority but with the objective to serve the external jobs as quickly as possible. Secondly, jobs that require execution at specific times (Reservation) are granted such times regardless of the consequences that will have in the remaining jobs. The simulation experiment was executed with trace files from the Parallel Workloads Archive (Feitelson 2005).	(i)**Multiple queues with no job priorities or reservation**: the method recorded a remarkable improvement in job slowdown and better average job slow down. (ii) **Performance under heavy load** When the arrival rate of jobs was increased to create an environment of heavy load, the multiple queue back-filling provided better average job slowdown than the single queue backfilling for all job classes. (iii)**Performance under reservation** Sets 0.01, 0.05 and 0.25 were used for each job input as proportions of jobs requiring reservation in this experiment. The multiple queue backfilling method showed better average job slowdown and a comparable slowdown for the 0.25 proportion set. In each of the experiments, the multiple queue back-filling method performance declined or gets worse for the long job class. This was because the queued jobs tend to compete with other jobs on the queue and shorter jobs tend to get scheduled more quickly than long jobs. The Multiple back-filling algorithm therefore favours smaller job

Fusion and Enhancement Algorithms		
Algorithm/Characteristics	**Simulation/scenario**	**Performance Result**
SCP(Set Covering Problem) - based heuristic Venugopal and Buyya (2008) Tasks are first matched to compute resources using: (i) Compute-first-where the computer resource that provides the least execution time is selected first. (ii) Exhaustive search- where all possible resource matching are generated and the one that guarantees the least completion time is chosen for the job. (iii) Greedy selection - in this case datasets are matched to compute resources through an iteration process. After each iteration, a check is made to compare it to the last iteration. After (i) to (iii) have been used to make the matching, the MinMin algorithm and Suffrage heuristic is applied. In the Suffrage heuristic, a resource is allocated to a job that will suffer the most if the compute resource was not allocated to it. The suffrage value is obtained by subtracting the second best CT value from the best CT value for the task.	GridSim was used to model the test bed containing 11 resources spread across 6 countries connected via high capacity network links. Each resource was used as both compute and data host except the one at CERN which was used for only data source. All resources were simulated as clusters of a single CPU node or processing elements (PE) with a batch job processing system using space shared policy. Each processing node or PE was rated in MIPS. Storage was modeled as total disk capacity at the site.Networks between links were modeled as routers and links. A uniform set of 1000 datasets was used for the evaluation and the set was distributed uniformly between 1GB and 6GB. The data were distributed uniformly and or Zipf-like. The degree of replication of data was set at 5. A bag of tasks that can be converted into a set of independent tasks was modeled. The size of application was determined by the number of jobs in the set (N). The size or length of each job is the time taken to run the job on a standard PE with MIPS rating of 1000. Each job requires a number of datasets selected at random from the dataset as input. 50 simulated experiments were conducted with different values for N, K, Size and Dist	As more jobs are submitted, the makespan for SCP and exhaustive search were lower compared to compute-first and greedy. (ii) Locality of access is higher for SCP and Exhaustive search as the number of jobs increases because as the number of jobs increases, there is the probability of accessing more jobs locally. The locality for Zipf-distribution is lower than the case for uniform distribution. When the number of datasets per job is increased the impact of data transfer time increased at a faster rate for greedy than for SCP and exhaustive search and the locality reduced steeply for Zipf-distribution. The effect of data transfer was reduced as the size of computation increased.

QoS Focused Algorithm		
Algorithm/Characteristics	**Simulation/scenario**	**Performance Result**
QoS Guided MinMin heuristic (QGMM) (He, Sun, and Laszewski 2003) Computes completion time of task, and host, then makes a match (best) between task and host for scheduling (minimum completion time over the entire host). Modification of MinMin with QoS matching as priority. Since smaller jobs always get completed before bigger jobs, this algorithm favours small jobs.	Simulated Grid Environment. Host parameter was fixed and three task submission scenarios: (a) 75% tasks need QoS requirement (network bandwidth of no less than 1.0 G bits/s). (b) 50% of tasks need QoS requirement. (c) Only 25% tasks need QoS requirements. Also, the frequency of scheduling for online mode, batch mode, MM and QMM were also compared. For each scenario and heuristics, 100 tasks were created 100 times and the makespan was computed separately.	Makespan of QGMM was better than MM in all the scenarios as specified below. (a) 8%. (b) 11.41% (c) 1.62%. Scheduling frequency in batch mode improved makespan for QMM(by approx.. 11 times) But for online mode, there was no difference.
AQuA- Availability-aware QoS Oriented Algorithm (Agarwaland Kumar 2011) Jobs or tasks are split into two parts (t1, t2). t1= tasks that require QoS (i.e. availability and bandwidth), t2= tasks that doesn't require QoS. Tasks in set t1 have higher priority and are scheduled to meet their QoS requirements before tasks in t2.	Results were validated in a simulated Grid environment. Results were compared against the (QGMM) (a) Grid size =100 nodes, tasks=1000 (percentage of dedicated nodes or availability of Grid resources was varied from 1 to 0.05 on a network of no less than 1Gbps. (b) Grid size varied from 50 to 1000 nodes. (c) Task size was varied over Grid resources. (Grid Size=100 nodes) (d) The percentage of tasks requiring QoS was varied and plotted against (i) Reliability and (ii) Makespan.	Reliability of Grid resources and makespan of tasks was used as performance metrics. (a) AQuA performed better as availability increased with better makespan and reliability (b) AQuA was more reliable with fewer jobs but with no effect in makespan. (c) AQuA was more reliable with increasing jobs with no effect on makespan. (d) AQuA was more reliable and with little better makespan.

QoS Focused Algorithm		
Algorithm/Characteristics	**Simulation/scenario**	**Performance Result**
NIMROD-G (Buyya, Abramson, and Giddy 2000 This model deals with ECONOMIC principle of SUPPLY and DEMAND. The model considered three key players in the GRID; (i) Grid Service Providers (GSPs) that represent the *producers*. (ii) Grid Service Brokers (GRBs) – that *represent brokers* and (iii) Grid Market Directory (GMD) which is the medium through which the two players in (i) and (ii) interact. The resource broker is made of: (i) task farming engine (ii) a schedule advisor and (iii) a dispatcher It uses the theory of supply and demand to match user tasks with Grid resources. QoS requirements of user jobs are used as conditions for a match. Matching is either Time constrained or Cost constrained.	The experiment was conducted on the WWG test bed. Deadline and budget constraints were considered. Experiments was conducted at two different times (Australian peak andoff-peak hours) on resources distributed in two major time zones using a "cost-optimization scheduling algorithm". The test bed has heterogeneous computer resources distributed across five continents: Asia, Australia, Europe, North America and South America. The test bed contains other resources as PCs, workstations, SMPs, Clusters, and vector supercomputers. The experiments were conducted based on: (i) Optimized for time (ii) Optimized for cost.	The broker selected resources in such a way that the whole application execution is completed at the earliest time for a given budget. (ii) The broker selected cheap resources to minimize the cost of execution and still try to meet deadlines. The experiment was really not compared against other Grid scheduling algorithms.

Adaptive Scheduling Algorithms		
Algorithm/Characteristics	**Simulation/scenario**	**Performance Result**
Resource Aware Scheduling Algorithm(RASA) Parsa and Entezari-Maleki (2009) Apply MinMin and MaxMin algorithms to schedule jobs. Implements MinMin and MaxMin in alternating sequence. If the number of jobs is ODD then it applies MinMin, and if the number is EVEN, then it applies MaxMin. The MinMin is favours smaller tasks while the MaxMinfavours larger jobs.	GridSim toolkit was used for simulating a Grid environment. Two assumptions were used: (i) the computation time of task overcomes communication time (ii) the communication time increases and even overcomes computation time of tasks. It was assumed that there are no constraints for executing tasks on different resources and each task could execute on each of the resources. Three different scenarios (workloads) were tested: Light= 200 tasks; Medium =1000 tasks; and Heavy=5000 tasks. Tasks were dispatched to 10 or 11 Grid resources. The tests were run against QGMM, Max-Min, and OLB. (a) Workload was increased from 17 to 725 based on assumption (i) (b) Workload was increased from 38 to 1186 based on assumption (ii)	(a) RASA returned the best (smallest) makespan based on assumption (i) and a small scale distribution of load. (b) RASA achieves smaller makespan even in a heavy workload situation.

Scheduling Algorithm based on Nature		
Algorithm/Characteristics	**Simulation/scenario**	**Performance Result**
Nature's Heuristics for Scheduling Jobs on Computational Grids (Abraham, Buyya and Nath2000) (i) Genetic Algorithm (GA). (ii) Simulated Annealing (SA). (iii) Tabu- Search (TS). (iv) GA-SA – Hybridization of GA and SA. (v) GA-TS – Hybridization of GA and TS. (i) GA: Uses optimization theory, theory of natural selection and survival of the fittest and adaptation. (ii) SA: This search is analogous to how metals cool and freeze into a crystalline structure. It is hoped that the process avoids ending up at any other point that is not optimal. (iii) TS: This search for solution method is aimed at starting off from one solution point and iteratively exploring neighborhoods for better solutions	(i) Jobs are allocated on FCFS basis and also LJFR, if a resource becomes free, further jobs are allocated on a SJFM bases. Thereafter, LJFR and SJFM are applied alternatively. After every job completion, apply the fitness test and apply mutation operation to get the optimum (user requirements). (ii) Hybrid GA-SA: jobs are allocated to available resources based on LJFM, once a resource becomes available due to job completion, a job is allocated based on SJFM and thereafter, LJFR-SJFR is applied after completion of every job, after every schedule, a mutation is carried out to replace old result with a better one. (iii) Hybrid GA-TS: A maximum number of feasible schedules are generated, then the makespan is evaluated for best schedule. Each best move made is counted and an Aspiration value is set, the next schedule will then begin from a neighborhood of the best value.	(i) A simulated experiment was carried out for only the GA algorithm with a finite number of resources (just 3 computing resources) and 13 jobs. An assumption was made that the processing speed of the resources and the cycles per unit time (CPUT) and the job length (processing requirements in cycles are known. The simulation showed that all the resources were efficiently utilized and the jobs were completed in minimum time. But only three resources and thirteen jobs is too minuscule to consider generalizing for the entire Grid (ii) No experimental tests or results were carried out for this experiment (iii) For this too, no experimental results was carried out.

Appendx D: Some Research that employed the MinMin Scheduling Algorithm for Comparison

S/No	Researchers	Algorithm/Title	Compared Against
1	Fujimoto, and Hagihara (2004)	Fujimoto, N., and Hagihara, K. (2004) 'A comparison among grid scheduling algorithms for independent coarse-grained tasks.' In *International Symposium on Applications and the Internet Workshops.* 674-680. IEEE	Compared the total processor cycle consumption (TCCP) of their proposed RR method with MinMin, MaxMin, WQ(work queue), DFPLTF(Dynamic Fastest Processor to Largest Task First) and Sufferage-C
2	Nesmachnow, and Canabe (2011)	Nesmachnow, S., and Canabé, M. (2011). GPU implementations of scheduling heuristics for heterogeneous computing environments. In *XVII Congreso Argentino de Ciencias de la Computación*	MinMin and Sufferage
3	Ye, Rao, and Li (2006)	Ye, G., Rao, R. and Li, M., (2006) 'A multiobjective resources scheduling approach based on genetic algorithms in grid environment'. In *Fifth International Conference on Grid and Cooperative Computing Workshops* 504-509 IEEE	Minin and MaxMin

S/No	Researchers	Algorithm/Title	Compared Against
4	He, Sun and Laszewski (2003)	He, X., Sun, X. and Laszewski, V. (2003) 'QoS guided min-min heuristic for grid task scheduling. *Journal of Computer Science and Technology*,*18*(4),442-451	MinMin
5	Wu, Shu and Zhang (2000)	Wu, M, Y., Shu, W., and Zhang, H. (2000) 'Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems. In *hcw* 375. IEEE	MinMin
6	Pinel, Dorronsoro and Bouvry(2012)	Pinel, F., Dorronsoro, B., and Bouvry, P. (2013) 'Solving very large instances of the scheduling of independent tasks problem on the GPU'. *Journal of Parallel and Distributed Computing*, *73*(1), 101-110.	GPU-parallelised version of MinMin
7	Nesmachnow, Cancela and Alba(2011)	Nesmachnow, S., Cancela, H., and Alba, E. (2012) 'A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling'.*Applied Soft Computing*, *12*(2), 626-639	MinMin and Sufferage

S/No	Researchers	Algorithm/Title	Compared Against
8	Hephzibah and Easwarakumar (2010)	Hephzibah, M, D, D., and Easwarakumar, K, S. (2010) 'A double MinMin algorithm for task metascheduler on hypercubic p2p grid systems'. *International Journal of Computer Science Issues, 7*(4), 8-18.	MinMin and MaxMin
9	Xie and Qin (2005)	Xie, T. and Qin, X. (2005) 'Enhancing security of real-time applications on grids through dynamic scheduling'. In *Job Scheduling Strategies for Parallel Processing* 219-237. Springer Berlin Heidelberg.	MinMin, Sufferage and Earliest Deadline First algorithm (*EDF)*
10	Yu and Yu (2009)	Yu, X., and Yu, X. (2009) 'A new grid computation-based Min-Min algorithm'. In *Sixth International Conference on Fuzzy Systems and Knowledge Discovery, (*1) 43-45 IEEE	MinMin
11	Amudha and Dhivyaprabha (2011)	Amudha, T., and Dhivyaprabha, T, T. (2011) 'Qos priority based scheduling algorithm and proposed framework for task scheduling in a grid environment'. In *International Conference on Recent Trends in Information Technology (ICRTIT),* 650-655 IEEE	MinMin, QoS guided weighted mean time min (QWMTM) and Max-Min heuristic algorithms

S/No	Researchers	Algorithm/Title	Compared Against
12	Hao, Liu, and Wen (2012)	Hao, Y., Liu, G., and Wen, N. (2012) 'An enhanced load balancing mechanism based on deadline control on GridSim'. *Future Generation Computer Systems*, *28*(4), 657-665	FPLTF, MinMin, max–min, and LBEGS
13	Carretero, and Xhafa (2006)	Carretero, J., and Xhafa, F. (2006) 'Use of genetic algorithms for scheduling jobs in large scale grid applications'. *Technological and Economic Development of Economy*, *12*(1), 11-17	MinMin *LJFR-SJFR* (Longest Job to Fastest Resource – Smallest Job to Fastest Resource)

Appendix E: Project Ethical Approval

REGISTRY RESEARCH UNIT

ETHICS REVIEW FEEDBACK FORM

(Review feedback should be completed within 10 working days)

Name of applicant: Abraham Goodhead .....

Faculty/School/Department: [Engineering & Computing] EC Computing

Research projecttitle: APPLICATION OF TRAFFIC-LIGHT CONTROL MECHANISM FOR GROUP-BASED MULTI-SCHEDULING IN GRID

Comments by the reviewer

1. Evaluation of the ethics of the proposal:
2. Evaluation of the participant information sheet and consent form:

3. Recommendation:
(Please indicate as appropriate and advise on any conditions. If there any conditions, the applicant will be required to resubmit his/her application and this will be sent to the same reviewer).

	Approved - no conditions attached
	Approved with minor conditions (no need to re-submit)
	Conditional upon the following – please use additional sheets if necessary (please re-submit application)
	Rejected for the following reason(s) – please use other side if necessary
X	Not required

Name of reviewer: Anonymous .....

Date: 07/03/2012 .....