**Soft Advance Reservation: a Resource Management
Technique for Handling Errors in User Estimated
Request Runtimes**

By

Phuong Luu Hoang, B. Eng.

A thesis submitted to the Faculty of Graduate and Postdoctoral
Affairs in partial fulfillment of the requirements for the degree of

Master of Applied Science

in

Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Faculty of Engineering

Carleton University

Ottawa, Ontario, Canada

September 2012

© Copyright 2012, Phuong Luu Hoang

Library and Archives
Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 978-0-494-93472-2

Our file Notre référence

ISBN: 978-0-494-93472-2

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

The runtime estimates submitted by users for Advance Reservation requests on grids and clouds are often inaccurate. This research presents a Soft Advance Reservation (SAR) approach for handling errors associated with such requests during resource allocation. This approach allows a predefined proportion of requests to miss their deadlines during matchmaking when resources are allocated to requests. The algorithms investigated selectively compress the user estimated request runtimes for handling requests with overestimated runtimes. The thesis focuses on using the algorithms with an Any-Schedulability criterion-based matchmaker that does not require an exact *a priori* knowledge of the local scheduling policy of the resource. The research shows how an improvement in system performance can be attained using the SAR approach as compared to the traditional approach which requires that all AR requests meet their deadlines. A simulation-based performance analysis of the algorithms is performed and the results are discussed.

Acknowledgements

First, I am grateful to Cistel Technology, Ottawa and the Ontario Centres of Excellence that provided financial support for this research. In addition, I would like to thank my supervisor, Professor Shikharesh Majumdar, for his guidance, continuous support, and encouragement during the course of my thesis. Working with him was a privilege for me. In observing him, I not only developed many useful skills, but also learned a lot about important work ethics. I truly believe that these lessons will benefit me greatly in my future endeavors. Second, I would like to thank Jose Orlando Melendez for his help in setting up the GridSim Toolkit and for allowing me to reuse part of his simulation program.

Finally, I would like to thank my parents, my brother, and my aunt. Their help and encouragement helped me get up and keep on going after many disappointments and setbacks. I am truly blessed to have all of you here in my life. Thank you.

Table of Contents

Abstract.....	iii
Acknowledgements	iv
List of Tables	xi
List of Figures.....	xii
List of Appendices.....	xix
List of Abbreviations and Symbols	xxi
1 Chapter: Introduction	1
1.1 Background	1
1.1.1 Resources	2
1.1.2 Deployment Model	3
1.1.3 Services	4
1.1.4 Components of Grids and Clouds	5
1.1.5 Workloads in Grids and Clouds	7
1.2 Resource Management on Grids and Clouds.....	8
1.2.1 Errors in User Estimation of Request Runtimes	9
1.2.2 A Priori Knowledge of Resource Scheduling Policies and the Any-Schedulability Criterion	9
1.3 Thesis Motivation	10

1.4 Thesis Goals.....	11
1.5 Thesis Contributions	12
1.6 Thesis Outline	13
2 Chapter: Related Works	14
2.1 Matchmaking and Scheduling in Grids and Clouds	14
2.2 Errors in User Estimation of Request Runtimes	18
2.3 Handling Error in User Estimation of Request Runtimes.....	19
2.3.1 Allowing Requests with Underestimated Runtimes	19
2.3.2 Aborting Requests with Underestimated Runtimes	23
3 Chapter: Soft Advance Reservation Based Matchmaking Algorithms	34
3.1 Assumptions.....	34
3.2 Matchmaking Algorithm with SAR (Algorithm 1)	35
3.2.1 Matchmaking of Requests.....	35
3.2.2 Handling Underestimation of Request Runtimes for Executing Requests	39
3.3 Algorithm 2 - A Variation of Algorithm 1.....	40
4 Chapter: Simulation Model	41
4.1 Performance Metrics	41
4.1.1 Probability of Blocking (P_b).....	41
4.1.2 Rate of Accepted requests (A)	41

4.1.3 Percentage of Work Rejected (W_R)	42
4.1.4 Resource Utilization (U)	42
4.1.5 Useful utilization (UU)	42
4.2 GridSim Toolkit	43
4.3 Simulation Entities	44
4.3.1 Source	45
4.3.2 Broker	45
4.3.3 Grid Information Service (GIS)	45
4.3.4 Resource	45
4.3.5 Resource Characteristics	46
4.3.6 Resource Scheduling Policy	47
4.3.7 Request (Gridlet)	48
4.4 System and Workload Parameters	49
4.4.1 Proportion of Allowable Missed Deadlines (p_{miss})	49
4.4.2 Margin (Δ)	50
4.4.3 Window Size (W)	51
4.4.4 Number of Resources (R)	51
4.4.5 Allocation Algorithm (A_p)	51
4.4.6 Resource Scheduling Policy (Sched_Policy)	52

4.4.7 Types of Workload	52
4.4.8 Workload Model	52
4.4.9 Model of Error in User Estimation of Request Runtime	54
4.4.10 Additional Workload Parameters.....	58
4.5 Accuracy of Results	59
5 Chapter: Performance Evaluation	60
5.1 Impact of Arrival Rate (λ).....	60
5.2 Impact of Time between the Arrival Time and the Earliest Start Time of AR Request.....	65
5.3 Impact of Laxity.....	68
5.4 Impact of the Window Size.....	70
5.5 Impact of Margin (Δ)	72
5.6 Impact of the Proportion of Allowable Missed Deadlines (p_{miss})	74
5.7 Impact of Number of Resources (R)	77
5.8 Impact of Error in User Estimation of Request Runtimes	80
5.8.1 Impact of Changing Parameters in Default Model of User Estimation Error..	80
5.8.2 Impact of Different Models of Error in User Estimation of Request Runtimes	92
5.9 Impact of Request Runtimes.....	100

5.9.1 Impact of Mean Request Runtimes.....	100
5.9.2 Impact of Having a Big Range for Request Runtimes.....	102
5.9.3 Impact of Variability in Request Runtimes	104
5.9.4 Impact of High Variability in Request Runtimes	104
5.10 Impact of the Resource Scheduling Policy	107
5.11 Impact of Allocation Policy	108
5.12 Discussion	109
6 Chapter: Conclusions	113
6.1 Summary	113
6.2 Insights into System Behaviour and Performance	115
6.2.1 Impact of Using Soft Advance Reservation.....	115
6.2.2 Comparison of Algorithm 1 and Algorithm 2.....	116
6.2.3 Impact of System and Workload Parameters on Performance	117
6.3 Future Research Directions.....	119
References	121
Appendix.....	130
Appendix A.....	130
A.1 Impact of T_s	130
A.2 Impact of Laxity (L).....	131

A.3 Impact of Window Size (W)	133
A.4 Impact of Margin (Δ)	134
A.5 Impact of the Proportion of Allowable Missed Deadline (p_{miss}).....	136
A.6 Impact of Number of Resources (R)	137
A.7 Impact of the Percentage of Requests with Runtimes Underestimated by Users	139
A.8 Impact of the Mean of Percentage of Error in User Estimation of Request Runtimes for Abnormally Terminated Requests	139
A.9 Impact of the Mean of the Percentage of Error in User Estimation of Request Runtimes for Normally Terminated Requests	143
A.10 Impact of the Coefficient of Variation of the Hyper-exponential Distribution	147
A.11 Impact of Different Models of Error in User Estimation of Request Runtimes	151
A.12 Impact of Mean Request Runtimes.....	154
A.13 Impact of Having a Big Runtimes Range	155
A.14 Impact of Variability in Request Runtimes.....	157
A.15 Impact of High Variability in Request Runtime	159
A.16 Impact of Scheduling Policy	160
A.17 Impact of Allocation Policy	169

List of Tables

Table 1: Default values of resource characteristics provided by GridSim	46
Table 2: Default values of gridlet characteristics provided by GridSim	49
Table 3: System parameters, algorithms, and scheduling policies	50
Table 4: Summary of the workload parameters	53
Table 5: Summary of models of errors associated with user estimation of request runtimes.....	55
Table 6: Parameter values for η	57
Table 7: Distribution details for abnormally terminated requests	85
Table 8: Distribution Details for Normally Terminated Requests.....	88

List of Figures

Figure 4.1: GridSim architecture adopted from [43]	44
Figure 5.1: Impact of arrival rate on P_b	61
Figure 5.2: Impact of arrival rate on A	62
Figure 5.3: Impact of arrival rate on W_R	63
Figure 5.4: Impact of arrival rate on U	64
Figure 5.5: Impact of arrival rate on UU	65
Figure 5.6: Impact of T_s on P_b	66
Figure 5.7: Impact T_s on U	68
Figure 5.8: Impact of L on P_b	69
Figure 5.9: Impact of L on U	70
Figure 5.10: Impact of W on P_b	71
Figure 5.11: Impact of W on U	72
Figure 5.12: Impact of Δ on P_b	73
Figure 5.13: Impact of Δ on U	74
Figure 5.14: Impact of p_{miss} on P_b	75
Figure 5.15: Impact of p_{miss} on U	76
Figure 5.16: Impact of R on P_b	77
Figure 5.17: Impact of R on U	79
Figure 5.18: Impact of P_U on P_b	81
Figure 5.19: Impact of P_U on W_R	83
Figure 5.20: Impact of P_U on U	83

Figure 5.21: Impact of P_U on UU	84
Figure 5.22: Impact of M_1 on P_b	86
Figure 5.23: Impact of M_2 on P_b	87
Figure 5.24: Impact of HM_1 on P_b	89
Figure 5.25: Impact of HM_2 on P_b	90
Figure 5.26: Impact of HM_2 on U	90
Figure 5.27: Impact of CoV_1 on P_b	92
Figure 5.28: Impact of CoV_2 on P_b	92
Figure 5.29: P_b of all the algorithms when Feitelson's model is used	94
Figure 5.30: P_b of all the algorithms when all requests have overestimated runtimes	94
Figure 5.31: P_b of all the algorithms when all requests have underestimated runtimes ...	95
Figure 5.32: U of all the algorithms when Feitelson's model is used.....	96
Figure 5.33: U of all the algorithms when all requests have overestimated runtimes	96
Figure 5.34: U of all the algorithms when all requests have underestimated runtimes	97
Figure 5.35: UU of all the algorithms when Feitelson's model is used.....	98
Figure 5.36: UU of all the algorithms when all requests have overestimated runtimes ...	99
Figure 5.37: UU of all the algorithms when all requests have underestimated runtimes .	99
Figure 5.38: Impact of S on P_b	100
Figure 5.39: Impact of S on U	101
Figure 5.40: Impact of having a big runtime range on P_b	103
Figure 5.41: Impact of having a big runtime range on U	103
Figure 5.42: Impact of V_f on P_b	105

Figure 5.43: Impact of CoV on P_b	106
Figure 5.44: Impact of CoV on W_R	107
Figure 5.45: Impact of CoV on U	107
Figure 5.46: Impact of scheduling policy on P_b for Algorithm 1	108
Figure 5.47: Impact of scheduling policy on P_b for Algorithm 2	109
Figure 5.48: Impact of allocation policy on P_b for Algorithm 1	110
Figure 5.49: Impact of allocation policy on P_b for Algorithm 2	110
Figure A-1: Impact of T_S on A	130
Figure A-2: Impact of T_S on W_R	130
Figure A-3: Impact of T_S on UU	131
Figure A-4: Impact of L on A	131
Figure A-5: Impact of L on W_R	132
Figure A-6: Impact of L on UU	132
Figure A-7: Impact of W on A	133
Figure A-8: Impact of W on W_R	133
Figure A-9: Impact of W on UU	134
Figure A-10: Impact of Δ on A	134
Figure A-11: Impact of Δ on W_R	135
Figure A-12: Impact of Δ on UU	135
Figure A-13: Impact of p_{miss} on A	136
Figure A-14: Impact of p_{miss} on W_R	136
Figure A-15: Impact of p_{miss} on UU	137

Figure A-16: Impact of R on A	137
Figure A-17: Impact of R on W_R	138
Figure A-18: Impact of R on UU	138
Figure A-19: Impact of P_U on A	139
Figure A-20: Impact of M_1 on A	139
Figure A-21: Impact of M_1 on W_R	140
Figure A-22: Impact of M_1 on UU	140
Figure A-23: Impact of M_1 on UU	141
Figure A-24: Impact of M_2 on A	141
Figure A-25: Impact of M_2 on W_R	142
Figure A-26: Impact of M_2 on U	142
Figure A-27: Impact of M_2 on UU	143
Figure A-28: Impact of HM_1 on A	143
Figure A-29: Impact of HM_1 on W_R	144
Figure A-30: Impact of HM_1 on U	144
Figure A-31: Impact of HM_1 on UU	145
Figure A-32: Impact of HM_2 on A	145
Figure A-33: Impact of HM_2 on W_R	146
Figure A-34: Impact of HM_2 on UU	146
Figure A-35: Impact of CoV_1 on A	147
Figure A-36: Impact of CoV_1 on W_R	147
Figure A-37: Impact of CoV_1 on U	148

Figure A-38: Impact of CoV_1 on UU	148
Figure A-39: Impact of CoV_2 on A	149
Figure A-40: Impact of CoV_2 on W_R	149
Figure A-41: Impact of CoV_2 on U	150
Figure A-42: Impact of CoV_2 on UU	150
Figure A-43: A values of all the algorithms when Feitelson's model is used	151
Figure A-44: A values of all the algorithms when all requests have overestimated runtimes.....	151
Figure A-45: A values of all the algorithms when all requests have underestimated runtimes.....	152
Figure A-46: W_R of all the algorithms when Feitelson's model is used.....	152
Figure A-47: W_R of all the algorithms when all requests have overestimated runtimes	153
Figure A-48: W_R of all the algorithms when all requests have underestimated runtimes	153
Figure A-49: Impact of S on A	154
Figure A-50: Impact of S on W_R	154
Figure A-51: Impact of S on UU	155
Figure A-52: Impact of having a big runtime range on A	155
Figure A-53: Impact of having a big runtime range on W_R	156
Figure A-54: Impact of having a big runtime range on UU	156
Figure A-55: Impact of V_f on A	157
Figure A-56: Impact of V_f on W_R	157

Figure A-57: Impact of V_f on U.....	158
Figure A-58: Impact of V_f on UU.....	158
Figure A-59: Impact of CoV on A.....	159
Figure A-60: Impact of CoV on UU.....	159
Figure A-61: Impact of scheduling policy on A for Algorithm 1.....	160
Figure A-62: Impact of scheduling policy on W_R for Algorithm 1.....	160
Figure A-63: Impact of scheduling policy on U for Algorithm 1.....	161
Figure A-64: Impact of scheduling policy on UU for Algorithm 1.....	161
Figure A-65: Impact of scheduling policy on A for Algorithm 2.....	162
Figure A-66: Impact of scheduling policy on W_R for Algorithm 2.....	162
Figure A-67: Impact of scheduling policy on U for Algorithm 2.....	163
Figure A-68: Impact of scheduling policy on UU for Algorithm 2.....	163
Figure A-69: Impact of scheduling policy on P_b for NE.....	164
Figure A-70: Impact of scheduling policy on A for NE.....	164
Figure A-71: Impact of scheduling policy on W_R for NE.....	165
Figure A-72: Impact of scheduling policy on U for NE.....	165
Figure A-73: Impact of scheduling policy on UU for NE.....	166
Figure A-74: Impact of scheduling policy on P_b for NM.....	166
Figure A-75: Impact of scheduling policy on A for NM.....	167
Figure A-76: Impact of scheduling policy on W_R for NM.....	167
Figure A-77: Impact of scheduling policy on U for NM.....	168
Figure A-78: Impact of scheduling policy on UU for NM.....	168

Figure A-79: Impact of allocation policy on A for Algorithm 1	169
Figure A-80: Impact of allocation policy on W_R for Algorithm 1	169
Figure A-81: Impact of allocation policy on U for Algorithm 1	170
Figure A-82: Impact of allocation policy on UU for Algorithm 1	170
Figure A-83: Impact of allocation policy on A for Algorithm 2	171
Figure A-84: Impact of allocation policy on W_R for Algorithm 2.....	171
Figure A-85: Impact of allocation policy on U for Algorithm 2	172
Figure A-86: Impact of allocation policy on UU for Algorithm 2	172
Figure A-87: Impact of allocation policy on P_b for NE	173
Figure A-88: Impact of allocation policy on A for NE.....	173
Figure A-89: Impact of allocation policy on W_R for NE	174
Figure A-90: Impact of allocation policy on U for NE.....	174
Figure A-91: Impact of allocation policy on UU for NE.....	175
Figure A-92: Impact of allocation policy on P_b for NM.....	175
Figure A-93: Impact of allocation policy on A for NM.....	176
Figure A-94: Impact of allocation policy on W_R for NM.....	176
Figure A-95: Impact of allocation policy on U for NM.....	177
Figure A-96: Impact of allocation policy on UU for NM.....	177

List of Appendices

Appendix A.....	130
A.1 Impact of T_s	130
A.2 Impact of Laxity (L).....	131
A.3 Impact of Window Size (W)	133
A.4 Impact of Margin (Δ)	134
A.5 Impact of the Proportion of Allowable Missed Deadline (p_{miss}).....	136
A.6 Impact of Number of Resources (R)	137
A.7 Impact of the Percentage of Requests with Runtimes Underestimated by Users	139
A.8 Impact of the Mean of Percentage of Error in User Estimation of Request Runtimes for Abnormally Terminated Requests	139
A.9 Impact of the Mean of the Percentage of Error in User Estimation of Request Runtimes for Normally Terminated Requests	143
A.10 Impact of the Coefficient of Variation of the Hyper-exponential Distribution	147
A.11 Impact of Different Models of Error in User Estimation of Request Runtimes	151
A.12 Impact of Mean Request Runtimes	154
A.13 Impact of Having a Big Runtimes Range	155

A.14 Impact of Variability in Request Runtimes.....	157
A.15 Impact of High Variability in Request Runtime	159
A.16 Impact of Scheduling Policy	160
A.17 Impact of Allocation Policy	169

List of Abbreviations and Symbols

Δ	Margin of p_{miss}
λ	Arrival rate
θ	Rate parameter
A	Rate of accepted requests performance metric
ACO	Ant Colony Optimization
A_p	Allocation algorithms
AR	Advance Reservation
AS	Any-Schedulability
CoV	Coefficient of Variation in hyper-exponentially distributed service time
EDF	Earliest deadline first scheduling policy
ESF	Earliest start time first scheduling policy
FF	First fit allocation policy
GASS	Grid Access to Secondary Storage
GIS	Grid Information Service
GRAM	Grid Resource Allocation Manager
GSI	Grid Security Infrastructure
IaaS	Infrastructure as a Service

L	Mean percentage laxity
MLF	Minimum Laxity First scheduling policy
NF	Next Fit allocation policy
OD	On-demand
PaaS	Platform as a Service
P_b	Probability of blocking performance metric
p_{miss}	Proportion of allowable missed deadline
P_U	Percentage of requests with underestimated runtimes
QoS	Quality of service
R	Number of resources
S	Mean request runtime
SAR	Soft Advance Reservation
SaaS	Software as a Service
Sched_Policy	Scheduling policy at the resource
SLA	Service Level Agreement
T_s	Time between the arrival of an AR request and its earliest start time
U	Resource Utilization performance metric

UU	Resource Useful Utilization performance metric
V_f	Variability factor in uniform distributed service time
VM	Virtual machine
VO	Virtual Organization
W	Window size
W_R	Percentage of work rejected performance metric

1 Chapter: Introduction

1.1 Background

Grids, and more recently clouds, provide a distributed platform for the execution of applications. A grid system is a collection of many resources that are not subject to centralized control and that use open-standard protocol for interaction between components. According to [16], there are three characteristics that define grid systems. A system is a grid if it

- “Coordinates resources that are not subjected to centralized control.”
- “Uses standard, open, and general-purpose protocols and interfaces.”
- “Delivers nontrivial quality of services.”

Cloud computing has recently gained more popularity as a platform for providing the computational resources on demand to users [31]. A Service Level Agreement (SLA) can also be associated with a user request. According to [46], cloud computing has become a prospective technology for both industrial and academic fields with many well known service providers such as Amazon, Microsoft, and Google.

There are many different definitions of cloud computing and researchers have not agreed on a single definition of cloud computing [11]. On the one hand, cloud computing can be considered as a computing model that derives from grid computing, public computing and Software as a Service (SaaS) according to the authors in [51]. Its defining characters are virtualization, distribution, and extendibility. On the other hand, many

researchers [11], [47] used the definition provided by the National Institute of Standard and Technology – NIST [37], which is presented here:

“Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.”

In grids and clouds, resources may come from different domains. Therefore, there are many challenges that need to be addressed which include security, policy, payment, and membership. In addition, the protocols and interfaces that address issues such as authentication, resource discovery, and resource management must be open and standardized to allow various types of resources to be integrated and used in these systems [31].

1.1.1 Resources

Grids and clouds can have multiple types of resources. The most common resource grids are data grids, computing grids, and storage grids [53]. Data grids share data resources. Computing grids make the sharing of computing power between computers possible. Storage grids allow users to access a large amount of storage space through the sharing of storage devices.

The currently available clouds offer computing and storage resources. Amazon is a well known cloud service provider that offers “Elastic Computing Cloud” (Amazon EC2 [54]) and “Simple Storage Service” (Amazon S3 [55]) cloud environments for users to buy computing cycles and storage on demand.

1.1.2 Deployment Model

A grid can be categorized based on the deployment of the corresponding resources [53] namely Local and Virtual Organization grids.

According to [17], a Virtual Organization (VO) grid is a set of individuals or organizations that engage in the sharing of resources. These individuals or organizations are seen as one large, distributed organization. Their resources connect with each other through the Internet. In this sense, resource sharing is not strictly confined to file exchange, but involves sharing of remote software, data, computers, and other resources. The sharing is highly controlled with clearly defined guidelines regarding things that can be shared, the individuals or organizations that are allowed to share and the conditions under which sharing occurs.

A local grid is a grid that is contained within a single organization or corporation and that can be accessed only by entities that are local or within the organization.

A cloud can be categorized in accordance with four different deployment models. They are private cloud, community cloud, public cloud, and hybrid cloud [37].

- *Private cloud* is an infrastructure that is used exclusively by a single organization. The organization may own and be responsible for operating and managing the cloud infrastructure. However, in some cases, a third party can manage the cloud on behalf of the organization.
- *Community cloud* is an infrastructure that is similar to a private cloud but it is shared and used exclusively by specific communities [11]. The responsibility

for owning and managing the cloud infrastructure may lie in one or more of the organizations that make up the community, a third party, or both.

- *Public cloud* is an infrastructure that is used by the general public on a subscription basis. A cloud provider such as business, academic, and government organization usually owns and is responsible for managing the cloud infrastructure.
- *Hybrid cloud* is a combination of more than one distinct cloud infrastructures described earlier. To enable the communication and application portability between different cloud infrastructures, standard or proprietary technology is used to bind these infrastructures together [37].

1.1.3 Services

Although grids are generally service independent, some can offer a specific type of service. For example, a render farm, used in rendering 3D animations, can offer its rendering service while using a specific rendering software package [53].

Clouds, in contrast, have three different service models that can be provided to the users. They are Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [11], [37]. SaaS is a service model in which the cloud provider's applications are accessible for the cloud users through a thin client such as Web browsers or a program interface. Users are typically charged on a pay-per-use basis. PaaS provides the tools to allow the developers to build and deploy web applications onto a cloud. Lastly, IaaS provides the storage and computing infrastructure to the users.

1.1.4 Components of Grids and Clouds

This section describes the most commonly used core components of grids and clouds.

1.1.4.1 Security

Security is an important aspect of both grids and clouds. Because they provide users with access to resources that may contain sensitive information, it is essential that the security of the information not be compromised by the system, according to [22]. The security component is used to ensure that an acceptable level of protection is maintained for all users. In the Globus Toolkit, the Grid Security Infrastructure (GSI) is responsible for security [23]. In the Amazon EC2 cloud, the Amazon EC2 Security component provides the required security services [25].

1.1.4.2 Matchmaking Component

This component, often called the “broker” [22], is responsible for determining the available and appropriate resources that can be used to serve user requests. The broker selects resources in order to ensure that a set of performance objectives are met [22].

1.1.4.3 Scheduling Component

This component is responsible for coordinating a set of requests to run on the resource. There may be different levels of schedulers within a system [22]. In systems that have clusters of resources, there may be a high-level scheduler (called a “meta-scheduler”), which schedules work onto a cluster, while the scheduler at the cluster is responsible for scheduling work onto the cluster’s individual nodes. The AppLeS (Application Level Scheduling) Parameter Sweep Template, described in [4] is an example of a

commercially available scheduler component that can be used in a grid and cloud environment.

1.1.4.4 Resource Management

Both grids and clouds comprise distributed resources that are not static. These resources can join and leave the system at any time. In order to manage these resources efficiently, a resource management component is required. Resource management comprises of four operators: resource discovery, resource scheduling, resource matching, and resource monitoring [50]. Within the Globus Toolkit, the Grid Resource Allocation Manager (GRAM) functions as the resource management component [23]. One example of the resource management component for clouds is an open source resource manager named Haizea [56].

1.1.4.5 Data Virtualization and Data Management

As user requests are executed, data may need to be made accessible to the node that is executing the requests. Since a grid and cloud system contains geographically distributed resources, data and computing power are often distributed among many locations. It is therefore necessary that access and management of distributed data be easy and effective, as though the data are stored in a central server [31]. The data virtualization and data management component provides these services. In the Globus Toolkit, the component is called the Grid Access to Secondary Storage (GASS) [23].

1.1.4.6 Charge-back

Businesses use charge-back to users based on their use of computing and data resources. The information can be used to charge users for using system resources. Utility grids and commercial cloud systems commonly use charge-back. The following are some examples of these systems:

- Amazon S3: Provides data-storage service.[55]
- Amazon EC2: Provides users with computational resources.[54]
- Google's App Engine service: Provides web-application developers with computing resources to run their applications.[57]

1.1.5 Workloads in Grids and Clouds

1.1.5.1 On-demand Requests

On-demand (OD) requests are the most basic type of requests. These requests are served on a best-effort basis. According to [42], most systems place incoming OD requests in a waiting queue if no resources are available to serve the requests. As a result, there is no guarantee of response time for these requests.

1.1.5.2 Advance Reservation Request

According to [13], [14], [40], [46], a QoS provision for an application is defined as an important requirement for grids and clouds. Because a system's resources are shared by many applications that are unaware of each other, it is difficult to provide a guaranteed response time for all these applications. For systems that have to provide users with a

QoS guarantee while maintaining effective resources utilization, Advance Reservation (AR) requests are beneficial.

According to [40], AR requests allow users to request a single or multiple resources at a specific time in the future. In addition, users can specify a deadline by which the request must be completed. Requests with Service Level Agreement (SLA) in clouds are often characterized by a deadline and are thus similar to ARs. As detailed in [32], each AR request is characterized by its earliest start time, deadline, and execution time. This ensures that all the required resources are simultaneously available for execution of the application at a specific time in the future. As a result, a QoS guarantee for the application can be provided.

In order to ensure effective utilization of resources, each AR request is accompanied by a laxity. A laxity is defined as the difference between the deadline and the time at which the request finishes executing given that it starts executing at its earliest start time. The laxity in the AR request gives the scheduler the freedom to order the requests in a way that best improves the performance of the resources.

1.2 Resource Management on Grids and Clouds

Matchmaking and scheduling on grids and clouds subjected to a workload comprising both ARs and ODs is a challenging problem. The problem is further complicated by the existence of uncertainties associated with user estimation of request runtimes and *a priori* knowledge of local scheduling policies used at the resources. Each of them is discussed in the forthcoming sections.

1.2.1 Errors in User Estimation of Request Runtimes

To perform matchmaking for ARs, the matchmaker requires users to estimate the runtime of a request at a resource. The estimates allow the matchmaker to determine which resource is most appropriate to serve the requests. To ensure that the matchmaker selects the most suitable resource, user-estimated request runtimes should be accurate. However, this is not always achieved in practice. In fact, a number of studies [1], [13], [14], [43], [45] point out that user-estimated request runtimes are highly inaccurate: users typically tend to overestimate request runtimes. There is also a small portion of requests with underestimated runtimes. The combined effect reduces the effectiveness of the matchmaker greatly [13], [14], [45]. Further discussion of the existing works on this is provided in Section 2.2 of Chapter 2.

1.2.2 A Priori Knowledge of Resource Scheduling Policies and the Any-Schedulability Criterion

The matchmaker typically needs to know the local scheduling policy at a resource to perform matchmaking for AR requests. In heterogeneous environments such as grids and clouds, it is not always possible to have this information *a priori*. There are two reasons for this. First, it is not always possible to know the exact configuration of grids and clouds at the time of system design. Second, the configuration is expected to change many times during the lifetime of the system. Even if the matchmaker knows the local scheduling policies of resources *a priori*, simulating the exact behaviour to check for the schedulability of a given set of AR requests at a resource is likely a high overhead operation.

The author in [29] introduces a novel approach to overcome this challenge. In [29], the author introduces the Any-Schedulability (AS) criterion. The AS criterion is a set of inequalities that determine if a set of ARs can meet their deadlines at a resource without knowing the resource's exact scheduling policy. The AS criterion is discussed in more detail in Section 2.1 (see Chapter 2).

1.3 Thesis Motivation

Using the AS criterion, the broker can perform matchmaking of requests to the resources without using *a priori* knowledge of the scheduling policy at these resources. However, it lacks the mechanism to deal with errors in user estimation of the request runtimes, which can negatively impact matchmaker performance. On one hand, as noted in [13], [14], requests with highly overestimated runtimes can lead to a low number of accepted requests on the system and can cause low utilization of resources. On the other hand, requests with underestimated runtimes waste the resources' time as these requests will get aborted in the future to avoid causing other scheduled requests to miss their deadlines. As a result, applying the AS criterion to the environment in the presence of user-estimation error will not be beneficial for resource providers unless there is a strategy that effectively deals with the error associated with user estimation of request runtimes.

Tackling errors associated with user estimation of request runtimes has been the focus of many researchers. Many approaches exist in the literature, such as [19], [24], [43], [44], that are effective in tackling the error in user estimation of request runtimes. These works try to improve the accuracy of predicting the request runtimes while ensuring that all requests meet their deadlines. Therefore, all requests with

underestimated runtimes are aborted. However, the authors in [14], [27], [48] show that the requirements of aborting these requests and ensuring all requests must meet their deadlines can be relaxed for achieving better performance. Indeed, the authors in [27], [48] show that allowing requests with underestimated runtimes to complete given that they do not delay other requests is beneficial to both the users and the system. In addition, the authors in [14] show that allowing a small number of requests to miss their deadlines can enhance the utilization of the system.

This research focuses on handling errors associated with user estimates of request runtimes in matchmaking and investigates the effectiveness of a matchmaking approach that allows a predefined proportion of requests to miss their deadlines and reducing the number of abortion of requests with underestimated runtimes. The approach utilizes the AS criterion and thus can be used even when detailed *a priori* knowledge of resource scheduling policies are not available.

1.4 Thesis Goals

This thesis introduces *Soft Advance Reservation* (SAR), described in Chapter 3, that is used in conjunction with selective compression of user estimates of request runtimes during matchmaking for handling errors associated with user estimates of request runtimes. The matchmaking strategy is based on the AS criterion. SAR allows a predefined proportion of deadlines for ARs submitted by a user to be missed.

Using simulation, the SAR approach is compared with the traditional AR approach where every AR must always meet its deadline. The performance of the SAR approach and related algorithms is evaluated through simulation. An in-depth analysis of

the performance of the SAR approach and the impacts of various system and workload parameters on system performance is performed.

1.5 Thesis Contributions

The major contributions of the thesis are summarized.

- Introduces an approach based on SAR and a selective compression of estimates of request runtimes for effectively tackling the problem of user-estimation error associated with request runtimes, thereby enhancing system performance.
- Introduces two matchmaking strategies based on the SAR approach that allows the broker to effectively perform matchmaking of requests in the presence of user estimation error associated with request runtimes. The algorithms are based on the AS criterion and can thus be used on systems on which a detailed *a priori* knowledge of the local scheduling policies used at resources may not be available.
- Presents a simulation-based performance analysis of the matchmaking strategies. The results of the simulation experiments demonstrate that allowing a certain proportion of requests to miss their deadlines along with a selective compression of estimates of request runtimes is an effective solution in the presence of errors associated with user estimation of request runtimes. Insights into the impact of various system and workload parameters on system behaviour and performance are presented.

1.6 Thesis Outline

This thesis is structured as follows. Chapter 2 provides a literature review of state-of-the-art in resource management and strategies in dealing with the user estimation problem. Chapter 3 describes the matchmaking strategies that use the SAR approach to matchmaking requests that have error in estimation of request runtimes. Chapter 4 details the simulation models and the simulation experiments that have been carried out to study the performance of the matchmaking strategies. Chapter 5 discusses the results of the experiments. Chapter 6 concludes the thesis.

2 Chapter: Related Works

This chapter describes the state of the art in matchmaking and scheduling algorithms in grids and clouds and algorithms that handle errors associated with user estimates of request runtimes. Matchmaking and scheduling algorithms that do not concern errors associated with user estimates of request runtimes are described in Section 2.1. A review of studies of errors associated with user estimates of request runtimes is discussed in Section 2.2. Finally, Section 2.3 provides a literature survey of matchmaking algorithms that can effectively handle error in user estimates of request runtimes.

2.1 Matchmaking and Scheduling in Grids and Clouds

Cloud computing is a promising technology where it allows users to solve complex problems by submitting their computing task to the cloud system. As users' demands of the cloud resources is increasing while the cloud resources are limited by their physical capacities, matchmaking and scheduling become more important in order to ensure the efficient use of cloud resources [39], [41]. A number of matchmaking and scheduling algorithms have been discussed in the literature. This section presents a representative set of them.

The author in [28] describes a matchmaking algorithm that is based on the Ant Colony Optimization (ACO) algorithm [10]. ACO is developed based on the foraging behavior of real ants that have been shown to be effective in finding an optimal path from food to nest [28]. The ACO-based matchmaking algorithm, called Load Balancing Ant Colony Optimization (LBACO), can allocate tasks so that their response times are minimized while system load is balanced.

LBACO assigns a virtual pheromone value to each virtual machine (VM). This value decays over time. When there is a task that needs to be allocated, an ant is placed randomly in every VM. Every ant visits other VMs randomly and tries to choose a VM that can serve the task. The probability of the ant choosing the VM for a task is higher if the pheromone value of that VM is high. A VM has high pheromone value if it is under light load and has the required speed to serve the task. When the ant finds a suitable VM for the task, it reinforces the pheromone value for all the VMs in the path that it has visited. In addition, if the path is the shortest path among other paths found by other ants, all VMs in that path are updated with higher pheromone values. This increases the probability that these VMs are considered for next tasks. The performance of the LBACO is studied using simulation. The experimental results show that the LBACO outperforms other matchmaking algorithms such as the First-come First-serve and the basic ACO.

The authors in [26] investigated a resource allocation architecture, called Topology-Resource Aware Allocation (TARA), that can be used in the IaaS cloud (see Section 1.1.3) to improve the performance of resource allocation. Using a set of predefined performance objective functions, TARA uses a prediction engine with a lightweight simulator to estimate the performance of a given resource allocation. It then uses a genetic algorithm to find an optimized resource allocation using the estimated performance provided by the prediction engine. TARA is prototyped to determine the performance of the proposed architecture. The prototype uses job completion time as an objective function. Using MapReduce [8] benchmark, TARA is shown to be able to

perform significantly better than allocation methods that are unaware of the applications submitted to the cloud.

The authors in [34] present a study of the performance of different gang scheduling policies in a simulated cloud system equipped with dynamic provisioning of VMs based on the current system's load. In the simulated system, each VM maintains a queue for tasks waiting to be executed on that VM. When there is an incoming job, the system dispatches the job's tasks into VMs if the number of tasks that belong to the job is less than the number of the system's available VMs.

For scheduling of tasks, two gang scheduling policies are used in their study. They are *Adaptive-First-Come-First-Serve* (AFCFS) and *Largest-Job-First-Serve* (LJFS).

When there is an idle VM, AFCFS goes through a list of ready-to-run jobs to find a suitable job for execution. A job is chosen for execution if two conditions are met. First, all of its tasks must be in front of their VMs' waiting queues. Second, these VMs must be idle. If there is no such job available, AFCFS looks for jobs whose tasks are further back in their queues in the same manner. This is done until it either finds a suitable job or no job is found.

With LJFS, jobs are scheduled depending on their size, which is the number of tasks belong to that job. Jobs with larger sizes are given higher priority than small jobs. The system maintains a list of *ready-to-run* jobs in descending order based on the jobs' sizes. When there is an idle VM, the system searches the list to determine which job to run. The first job, whose tasks are allocated to VMs which are idle, is scheduled for

execution. Simulation results presented in [34] prove that these gang scheduling policies can be efficiently applied to a cloud system with dynamic provisioning of VMs.

In [29], the author introduces a novel criterion, called Any-Schedulability (AS) criterion, which allows the matchmaker to determine if a set of ARs can meet its deadline at a resource without knowing the resource's exact scheduling policy *a priori*. The only assumption required by the criterion is that the scheduling policy be work-conserving, which requires the resource to serve any ready-to-start requests when there is an idle slot in the schedule. The criterion is described below. Each AR request i is characterized by the following properties.

S_i : Earliest start time for request i

D_i : Deadline associated with request i

E_i : Service time for request i

L_i : Laxity associated with request i ; defined as $L_i = D_i - S_i - E_i$

A given set of AR requests (where N represents the size of the set) will be scheduled successfully at a single resource if the following condition is satisfied for each request i ($i = 1 \dots N$).

$$L_i \geq \sum_{j \neq i} X_1 \times \{E_j, (D_j - S_i)\} + X_2 \times E_j$$

Where,

$X_1 = 1$ if any of the following is true

$$(S_j < S_i, D_j < D_i, S_j < D_i, S_i < D_j)$$

Or

$$S_j \geq S_i, D_j < D_i, S_j < D_i, S_i < D_j$$

Or

$$(S_j \geq S_i, D_j \geq D_i, S_j < D_i, S_i < D_j)$$

For other cases, $X_1 = 0$

$X_2 = 1$ if $(S_j < S_i, D_j \geq D_i, S_j < D_i, S_i < D_j)$ and $X_2 = 0$ for other cases

In other words, a set of requests is schedulable using any work-conserving scheduling policy if the laxity of every request i in the set is bigger than or equal to the total of worst case delays caused by all the requests in the set that overlap with the request i .

2.2 Errors in User Estimation of Request Runtimes

For ARs, users have to provide estimates for the runtimes required for requests associated with the request to run on the resource. The estimated runtime is used by the schedulers to order the requests as they run at the resources. A number of studies [1], [6], [13], [14], [43] have shown that users typically do not provide accurate estimation of request runtimes. Users tend to overestimate the required runtime, meaning that their estimates are greater than the amount of time that the requests actually need. One explanation for this, provided by Lee et al. in [1] states that users overestimate requests to prevent their requests from being aborted prematurely. The overestimation is grossly large. In fact, [1] reported that, on average, the majority of requests use less than 20% of the requested time. Another work [6] found that, in some workload traces, 35% of requests used less than 10% of their requested time. According to [13], [14], in some extreme cases, users can overestimate the time required by as much as 25,000%. In addition, users are typically confident in their estimates and cannot provide a better

estimate even if there are incentives to do so. The improvement of estimation accuracy resulting from incentives, as reported in [1], is observed to be marginal (with inaccuracy decreasing from 61% to 57%).

Typically, only a small portion of requests are underestimated, meaning that users estimate the runtime to be shorter than the amount of time that the request actually needs. In [13], [14], the authors analyzed the IBM workload traces and found that 2% of the requests were underestimated by the users.

2.3 Handling Error in User Estimation of Request Runtimes

Many approaches tackling errors in user estimation of request runtimes have been proposed in the literature. These approaches can be categorized into two types. The first tolerates requests with underestimated runtimes if they occur. Section 2.3.1 describes algorithms of this type. The second type of approach aborts these requests. Section 2.3.2 describes algorithms of this type.

2.3.1 Allowing Requests with Underestimated Runtimes

In this category, algorithms allow requests with underestimated runtimes to run to completion. However, the algorithms can be classified into different subcategories based on the type of requests with underestimated runtimes allowed to run to completion. The first subcategory allows only requests with underestimated runtimes generated by the system to run to completion. Section 2.3.1.1 describes these algorithms in detail. The second subcategory allows all requests with underestimated runtimes to run to completion. Section 2.3.1.2 describes algorithms in this subcategory in greater detail.

2.3.1.1 Allowing only Requests with Runtimes Underestimated by the System

The authors in [45] describe a way to tackle the negative effects of error in user estimation of request runtimes in the context of backfilling algorithms. This is done in two steps. First, the algorithm calculates a predicted runtime of the request, called system estimate of request runtime, and uses that value for scheduling. The authors in [45] showed that recent data is more important than historical data in predicting request runtime. Instead of mining the history for similar requests, the authors used the average runtime of the previous two requests submitted by the same user as the runtime predictor for the next request. If the average runtime turns out to be less than the user estimate, the average runtime is used. If there were no previous requests, the user estimation of the request runtime is used. The second step addresses what to do when the system estimate of request runtime is wrong. If the prediction of the request runtime is too short, it is extended. The user-estimated runtime serves as the kill-time, which is the time at which the request is aborted. The request is allowed to run until it finishes or until it reaches the user-estimated runtime. The authors showed that, while the algorithm is simple, it improves performance significantly.

An algorithm to allocate bag-of-tasks applications into the clouds is described in [38]. The algorithm seeks to allocate the tasks submitted by users to allow them to be executed within the specified budget the users are willing to spend, while minimizing the completion times. The algorithm does not need to know the runtime of a task *a priori*, but estimates the task's completion time in two phases: the initial sampling phase and the monitoring and feedback phase.

In the initial sampling phase, a set of sample tasks from the running tasks is selected for determining the times the tasks run at each resource. These tasks are allowed to run to completion at the resources and the algorithm maintains an ordered list of execution times for these finished tasks. After all the sample tasks have finished, the estimated runtime of a task i , denoted as T_i , running on the cloud resources is calculated using the cumulative moving average of the execution times of all the finished tasks in the sample set. The value of T_i indicates the amount of time a task in the bag will take to run at the resource. This information allows the algorithm to determine a set of resources that can serve the bag of tasks while ensuring that the cost incurred for using the resources is within the user's specified budget.

As more tasks complete, the estimated runtime of the task needs to be refined. This is done in the monitoring and feedback phase. The algorithm selects a short monitoring interval, which allows the value of T_i to be refined as more tasks run to completion. In addition, the use of a short monitoring interval makes it possible to detect budget violation. As a result, more machines can be allocated to prevent violation of the budget specified by the users.

The authors in [13], [14] showed that the existence of overestimation error in a request's estimated runtime in the real traces collected from the IBM SP2 supercomputer is extremely large. This leads to a large and unnecessary rejection of requests since the feasibility analysis at the time of request admission relies on an estimation of request runtime provided by the user. As a result, performance is severely degraded. In order to combat the issue of inaccuracies in user estimation, the authors used the pre-scheduling

mechanism, which allows over-booking of the system with requests. The system can allow a small proportion of requests to miss their deadlines. This is called the overbooking threshold (OBT). When a new request comes in, the system performs a schedulability analysis. If the analysis shows that accepting the request does not cause the proportion of missed deadline requests to be greater than the OBT, the system is allowed to accept the request. As most requests have overestimated runtimes, the actual number of requests that miss their deadline is lower than the OBT. According to [14], allowing a small proportion of requests to miss deadline improves the utilization of the system.

2.3.1.2 Allowing All Requests with Underestimated Runtimes

The authors in [48] show that requests with underestimated runtimes do not always have to be aborted. In contrast, they suggest aborting these requests only when they cause other requests to be delayed. Their experimental results show that allowing these requests to run longer than their estimated runtimes causes only a small degradation in system performance. In addition, it reduces the number of aborted requests significantly.

The authors in [27] describe a modified backfilling algorithm that does not abort requests with underestimated runtimes. In the algorithm, a request is allowed to run up to the estimated runtime provided by the user. If the request does not finish, it is allowed to run for an extended period as long as it does not cause other requests to be delayed. If extending the request causes other requests to be delayed, the request is aborted. The experimental results show that the modified algorithm outperforms the standard backfilling algorithm, which aborts all requests with underestimated runtimes. In addition, users have less incentive to overestimate the runtimes of their requests as

requests with underestimated runtimes are tolerated by the system. As a result, incentives for users who provide accurate estimated runtime for the requests become more attractive. As users provide more accurate estimates, system performance increases.

2.3.2 Aborting Requests with Underestimated Runtimes

In this category, the algorithms abort requests with underestimated runtimes when their elapsed times reach their estimated runtimes. This is done to prevent the possibility of delaying other requests in the schedule. The algorithms in this category try to provide a good estimate for a request's runtime. They can be classified into four subcategories based on the approach they take in providing an estimate: the overbooking approach, history-based approach, code analysis, and resource mapping.

The overbooking approach allocates more requests to a resource than the resource can actually serve. This is done because most request runtimes are overestimated by users and thus do not use the entire time requested. Section 2.3.2.1 describes algorithms in this subcategory in detail. The history-based approach utilizes the information from previous finished requests to predict the runtime of the next request [35]. Section 2.3.2.2 describes algorithms in this subcategory in greater detail. The resource mapping approach predicts a request's runtime using the characteristics of the resources that execute the requests. Section 2.3.2.3 describes some algorithms in this subcategory. The code analysis approach looks at the code of the application submitted to the systems to determine its runtime. Section 2.3.2.4 describes algorithms of this type in greater detail.

2.3.2.1 The Overbooking Approach

To resolve the problem of overestimation of request runtimes by users, the authors in [2] introduce an approach that uses overbooking. In this approach, incoming requests are placed in a free slot in the resource schedule even if the time slot is smaller than the estimated runtime of the request. A request is allowed to be placed in a free slot in the resource' schedule if the probability of success for that request on the resource is higher than a predefined value. The probability of success is calculated using two probabilities: the probability of resource failure and the probability of a request finishing on time if it is allocated to that resource. Using simulation, the authors show that overbooking allows resource providers to gain an additional 20% of resource utilization.

Another approach, called Relaxed Advance Reservation Policy (RARP), is introduced and investigated by the authors in [49]. In RARP, the authors assume that AR requests do not have any laxity. In other words, each AR request is associated with a definite start time and a hard deadline that it has to meet. RARP allows an AR request to overlap with existing AR requests at the resource if the calculated probability of violating the request deadline is less than a predefined threshold value. According to the authors, because request runtimes are usually overestimated by users, deadline violation is not likely to occur. Using simulation, the authors show that RARP reduces the number of requests unnecessarily rejected due to overestimation of request runtimes. However, this approach incurs a high number of SLA violations because resources are not able to serve all the requests if the threshold value is not chosen properly.

2.3.2.2 History-Based Approach

The authors in [36] introduce a matchmaking algorithm that aims to reduce the response time of the requests. It uses the history of past requests' runtimes to predict the total runtimes of the requests queuing at each resource. This information allows the matchmaker to determine which resource is freed the earliest. When a request comes, the matchmaker is able to use the information to allocate the request to the resource where it can start the earliest.

In order to predict a request's runtime, the matchmaker retrieves the runtime of past request in the requests history which has similar command names as specified in the request. If it cannot find a request with a similar command name, the predicted runtime of the request is computed by calculating the average runtimes of all the requests in the requests history.

In [30], the authors assess the effectiveness of different machine-learning algorithms in predicting the request runtimes in grid and cloud systems. In addition, they devised an algorithm called Predicting Query Runtime (PQR2), which can achieve a high accuracy in predicting request runtimes.

A heterogeneous environment consisting of four different hardware resources is set up to test the performance of the algorithm. Two bioinformatics applications are set to run in this environment. The accuracy of the devised algorithm in predicting the runtime of these applications is compared against other machine-learning algorithms. The experimental results show that the devised algorithm achieves the highest accuracy of all other algorithms in predicting the runtime of these applications.

The authors in [52] provide a performance model that can be used to estimate the execution time of a Pig program [18] for a given resource allocation in the clouds. Since a Pig program can be compiled into an execution plan containing many MapReduce [8] jobs, the authors aim to provide accurate estimate of the MapReduce jobs belonging to the Pig program.

The execution of a MapReduce job can be divided into two phases: *map* and *reduce* phases. Each phase contains a set of tasks that can be executed in parallel. According to [52], the upper bound and lower bound on the execution times of each phase can be derived from the predicted average and maximum execution times of all the tasks in that phase. The predicted average and maximum execution times can be calculated using historical data on the execution times of previous tasks. The lower bound and upper bound on the execution times of the job can then be calculated using the bounds on execution times for all the job's phases. According to the authors in [52], the average of the lower bound and upper bound on the execution time of the job is usually a good indication of the job's execution time. The prediction of the total execution time of the Pig program can be derived from the execution times of all the MapReduce jobs.

An approach that utilizes historical information about user-estimation accuracy in the context of backfilling algorithm is described in [43]. The authors studied the inaccuracies of user-estimation based on real traces collected in Blue Gene/P systems. The authors showed that user estimation is highly erroneous in these systems. Moreover, more accurate estimation of request runtimes leads to a better overall performance. Based on their observations, they devised a runtime adjustment scheme using the history of

user-estimation accuracy. The user-estimation accuracy, denoted by R , is defined as the ratio of the request's actual runtime over the user estimation of the request runtime. Higher the value of R , more accurate the user estimation of the request runtime is. This value is usually less than one since users tend to overestimate the request runtimes to prevent their requests aborted before completion. The use of user-estimation accuracy is argued to provide more accurate runtime estimates [43]. In order to calculate the accuracy, the authors used four different schemes: a user-based scheme, a project-based scheme, a combined scheme, and a selective scheme.

- In the user-based scheme, requests history from the same user is used to calculate the predicted user-estimation accuracy, denoted as A , which is then used for predicting the request runtimes. If the number of requests from the same user in the historical workload is greater than a chosen threshold value, denoted by N , the value of A is set as 80th-percentile of these requests' R values [43]. In other words, A is larger than 80% of these requests' R values. If the number of requests in the historical workload is less than N , A is set to value of 1 to keep the predicted request runtimes to be similar to the user estimation of the request runtimes due to a lack of historical data.
- In the project-based scheme, request information from the same project is used to calculate A and predict the request runtimes. The calculation is similar to user-based scheme.
- The combined scheme is a combination of the user-based and project-based schemes. It uses the information from both the project and user if they both

have A values different from 1 [43]. If either scheme has A equal to 1, the value of A for the other scheme is used. Otherwise, the runtime is not adjusted due to a lack of historical data.

- The selective scheme performs runtime adjustment on certain kinds of requests. In order to achieve accurate request ordering in the schedule and efficient selection of requests to be backfilled to utilize the free resource slot, the adjusted estimated runtime of the request is used. For reserving a time slot in the resources for ensuring the request at the top of the queue can run when it is ready to execute, the user-estimated runtime was used to increase the chance of backfilling. As request runtimes are usually overestimated by the users, relying on the user-estimated runtimes of the requests means that these requests may finish before the reserved time at which the high priority request can start. This creates free time slots in the resources that other low priority requests can backfill.

The user-based scheme performs the worst. This is because the historical data for each user is not as rich as for the project. This reduces the number of requests that have their user estimation of request runtimes adjusted. The project-based scheme has more information and thus performs better than the user-based scheme. Because the combined scheme utilizes both the project-based and user-based schemes, it outperforms both individual schemes. The selective scheme also shows better results than the project-based and user-based schemes. However, it is not as good as the combined scheme in terms of the entire average.

2.3.2.3 Resource Mapping

One approach that utilizes information about a resource to enhance the prediction of requests' runtimes is described in [44]. This approach is an algorithm for performing meta-scheduling in advance in grid. Meta-scheduling algorithm, as described in [44], is the first step in the process of reserving a resource in advance. When there is a request for using resources in grid, this algorithm selects the resources and the time period in which the request will be executed. The system keeps track of the selection information. However, these resources are not physically reserved. Instead, the selection information is used later when it is possible to physically reserve the resource for the request.

The authors' main contribution is the implementation of a framework for network-aware meta-scheduling in advance algorithm. The framework is built on top of Globus and the GridWay meta-scheduler. In the framework, there are three approaches to predicting request runtimes. These approaches utilize historical data about requests completed at the resources to make the prediction.

The first approach, called Total Completion Time (TCT), calculates the total completion time of request at each resource using the historical data about previous execution times of requests at that resource. In order to reduce the overhead in calculating the completion time of the request, this approach only selects resources having free time slots overlapping with the period between the request's earliest start time and deadline for computation of the completion time. For each of these resources, the average execution time for all executed requests is calculated. This average execution time is used as the predicted execution time of the request if it is allowed to run at the resource. In this

approach, the transfer time of the request from the user machine to the resources is considered together with the predicted execution time of the request.

The second approach differs from the first in that it predicts a request's execution time at the resource and the transfer time separately. This approach is called Execution and Transfer Times Separately (ETTS). The execution time for the request at the resource is predicted in the same way as in the first approach. The transfer time includes the transfer time for the input and output files. First, the available bandwidth of the network to be used is predicted by calculating the mean bandwidth used on the previous day during the same time interval that the request will be allocated in. The input/output transfer time is then calculated by dividing the size of the transfer file with the mean bandwidth. The total execution time of the request is the sum of the estimated execution time at the resource, the predicted transfer time for the input file, and the predicted transfer time for the output file.

The third approach is an extension of the second approach. In the paper, it is referred to as Resource Trust (RT). It predicts a request's execution time in a similar way as the second approach. However, the predicted execution time is fine-tuned using the resource trust. The resource trust for a request i , denoted as $RT(R_i)$, is defined as the average error made for the last N requests that finished. A negative value of $RT(R_i)$ indicates that the estimated execution time is less than the real execution time. In this situation, the approach requires the calculated execution time to be extended. The added time is the absolute value of the resource trust. This ensures that the job has a greater chance of finishing.

The three approaches were studied in a local real grid environment. The experiments showed that both ETTS and RT outperform TCT, which indicates the need for estimating the transfer time and the execution time separately. Furthermore, using RT to fine-tune the predicted time clearly improves the QoS perceived by the users.

In [5], the authors investigated a resource-mapping approach for predicting the runtime of the requests in computational grid. The prototype subsystem comprises three components: the Resource Monitor, the Resource Predictor, and the Runtime Predictor. The Resource Monitor periodically collects information about the average host load of the resource. The average host load is measured since the runtime for the compute-bound job is strongly related to the average host load it experiences during its execution, as described in [9]. The Resource Predictor uses the collected measurements to predict the future usage of the resource. It employs a linear time series model to predict the load of the host resource. The Runtime Predictor then uses the predicted load value to predict the runtime for the requests that execute at the resource. It uses the runtime of the request on an unloaded resource and the predicted load as the inputs for calculating the runtime of the requests on the loaded resource. The experiment showed that the relative error is acceptable. For example, a 6.7% relative error was achieved when the unloaded runtime was 50 seconds. In addition, the overhead for this algorithm is observed to be small.

2.3.2.4 Code Analysis

The authors in [24] describe a prediction module for estimating the execution time of an application submitted to the grid. The idea is that the submitted application goes through four layers of examination before a runtime prediction can be made. The first layer, called

the “Application Selector” decides which software was used to develop the submitted application. That knowledge is needed to load the appropriate “File Parser.” The second layer, the File Parser, is composed of three components: “Lexical Tokenization”, “Keyword Counter”, and “Grammar Parser”. The Lexical Tokenization component performs lexical analysis of the program. Keyword Counter determines the number of basic operators, such as subtraction, addition, etc., and keeps track of a list of functions and program iterations discovered in the program. The Grammar Parser component parses the program using a predefined set of rules to enable the prediction module to better understand the structures of the conditional and iteration statements. The information obtained by the File Parser is then provided to the “Code Evaluation Engine” at the third layer. The Code Evaluation Engine determines the complexity of the application in terms of the number of lines of codes, the number of distinct operators, and operands. These pieces of information are used by the “Predictor Engine” to predict the time required to execute the entire application. The Predictor Engine makes use of the tokens from the second layer as well as the benchmark data available for these tokens to determine the total execution time of the application. The benchmark data contain the execution times for commonly used operators and functions.

The performance of the execution time prediction module developed for predicting the execution of programs written using R programming language was studied in 60 different test cases. The R programming language is used to develop scripts that can be run inside R! application, which is used for performing statistical and graphical

computations. The test results showed that in 44 out of 66 tests, the estimation accuracy was more than 80%.

However, there are several drawbacks to this solution. First, it requires the source code of the application to be known, which may not be possible in some cases. Second, the parser for the programming language used to write the application has to be available before prediction can be made. This means that predictions for applications using a new programming language cannot be made due to not having the execution time prediction module available for the language. Finally, even though this solution does not require historical data of requests to be available, the benchmark data must be available for the machines on which the requests will be run.

A number of techniques for handling error in user estimation of request runtimes are discussed in this chapter. This thesis investigates an approach based on SAR and selective compression of estimates of request runtimes for tackling the problem of handling errors associated with user estimation of request runtimes. By utilizing the Any-Schedulability criterion, the SAR based matchmaking algorithms can be deployed on systems on which exact *a priori* knowledge of scheduling policies used at resources may not be available. None of the existing works presented concern such matchmaking algorithms that allow a predefined proportion of requests to miss their deadlines and can be used without having exact *a priori* knowledge of resource scheduling policies that this thesis focuses on.

3 Chapter: Soft Advance Reservation Based Matchmaking Algorithms

This chapter describes two algorithms that are based on Soft Advance Reservation (SAR). With a SAR, a certain predefined proportion of the requests are allowed to miss their deadlines. The proportion of requests that are allowed to miss their deadlines is denoted by p_{miss} . However, requests that miss their deadlines due to underestimation performed by the user are not included in the computing of p_{miss} .

This thesis implements two strategies that employ SAR. The first one, called Algorithm 1, is presented in Section 3.2. Another variation of Algorithm 1, called Algorithm 2, is described in Section 3.3.

3.1 Assumptions

The following section describes the assumptions made in the development of the algorithms:

- The requests are independent of each other.
- The only types of requests arriving to the system are AR requests. The AS criterion described in [32] is used in matchmaking.
- Resources considered in this thesis are homogeneous and have the same speed. On systems in which resources may have different speeds, the estimated runtimes of the requests can be scaled appropriately to correspond to the speed of the resources.
- The transfer time (for requests to reach the resources) is negligibly small compared to the request's execution time at the resource.

- All requests are pre-emptable. Preemption overhead is assumed to be negligibly small in comparison to the mean request runtimes.
- The matchmaking decision is made at the resource broker. The matchmaking algorithm utilizes the AS criterion that works with a non-shared resource model: each resource serves only one request at a time.
- The local scheduling policy used at the resource follows the work-conserving principle. This principle states that a CPU should never be idle when there is a ready request that can be executed on the resource. Many existing scheduling policies follow this principle [32].

3.2 Matchmaking Algorithm with SAR (Algorithm 1)

Algorithm 1 performs two main operations. First, it performs the matchmaking of requests to resources, described in detail in Section 3.2.1. Second, it determines how requests with underestimated runtimes are to be handled by the system. Section 3.2.2 has a description on how this process is performed.

3.2.1 Matchmaking of Requests

The matchmaking process has five steps. The flow of the steps is described below.

Step 1: The matchmaker decides whether the user estimate of request runtime or a predicted runtime of the incoming request generated by the system should be used for matchmaking for that request. The details of this step are provided in Section 3.2.1.1.

Step 2: If the predicted runtime of the request is used, the flow continues to step 3. If the user estimated runtime of the request is deemed appropriate for matchmaking, the flow continues to step 4.

Step 3: The predicted runtime of the request is calculated using the information from previous requests. The details of this step are provided in Section 3.2.1.2.

Step 4: The matchmaker utilizes the AS criterion presented in detail in Section 2.1, to determine the resources that can serve the request.

Step 5: If there are multiple candidate resources, the matchmaker selects the one using an allocation algorithm. The allocation algorithms that can be employed by the matchmaker are presented in Section 3.2.1.3.

3.2.1.1 Determining the Estimated Runtime of the Request

When the broker has to perform matchmaking for a request, it has to determine whether or not to use the system estimate of request runtime. Since the system estimate of request runtime may not always be a good indicator of request runtime, it is applied to requests only when it is safe to do so, i.e. when Inequality 3.1 is satisfied.

The system keeps track of the ratio of requests that missed their deadlines so far, denoted by R_{miss} . R_{miss} is the ratio between number of missed deadlines for requests caused by the matchmaker using the system estimate of the request runtime and the number of requests finished and aborted so far. A missed deadline for a request is considered to be caused by the matchmaker if it has not finished and its total runtime on

the resource so far is less than the estimated runtime provided by the user at the time its deadline is reached.

When there is a request that needs to be allocated to a resource, the matchmaker uses the system estimate of the request runtime for matchmaking as long as Inequality 3.1 is satisfied.

$$R_{miss} \leq (p_{miss} - \Delta) \quad (3.1)$$

In other words, the system uses its estimate of the request runtime if R_{miss} is less than the value of p_{miss} reduced by Δ . In Inequality 3.1, Δ , called margin, is a threshold value used to prevent R_{miss} from accidentally exceeding the value of p_{miss} . For example, if p_{miss} is 10% and Δ is 5%, R_{miss} is only allowed to increase up to $p_{miss} - \Delta = 5\%$. As the difference between p_{miss} and R_{miss} is 5%, this ensures SLA of the users is not accidentally violated.

If Inequality 3.1 is not satisfied, the user-estimated runtime of the request is used. This is done to avoid risking having more requests missing their deadlines and the value of R_{miss} exceeding p_{miss} which can cause user dissatisfaction due to having more requests missing their deadlines than previously agreed.

3.2.1.2 Calculating System Estimate of Request Runtime

This section describes the details about how to *compute an estimate of request runtime for matchmaking purposes*. The result of computation is called system estimate of the request runtime estimation, in contrast to the estimation of the request runtime provided by the users.

System estimate of request runtime are often used for matchmaking purposes. To calculate the *system estimate of request runtime*, a history-based approach is adopted. This approach uses the statistics of previous requests executed on the system and does not need a training session based on a training set of requests that are representative of the system workload. The system keeps track of W previously executed requests. The matchmaker utilizes this information to predict the runtime for the current request. The information utilized by this algorithm is the percentage of overestimated runtime for previously finished requests (α). α is used to reduce the user-estimated runtime of the incoming requests. Since requests with underestimated runtimes caused by the users have actual runtimes equal to their user estimated runtimes, they are not used in the calculation of α . The value of α is calculated as follows:

$$\alpha = \frac{\sum_{k=1}^W O_k}{W} \quad (3.2)$$

Where O_k is the percentage of overestimated runtime for request k and is calculated as follows:

$$O_k = \left(\frac{T_E - T_R}{T_E} \right) \times 100 \quad (3.3)$$

In the equation above, T_E is the estimation of request runtime provided by the user. T_R is the actual runtime of the request at the resource. The system estimate of the runtime for the incoming request, T_M , is calculated as follows:

$$T_M = T_E \times \left(1 - \frac{\alpha}{100} \right) \quad (3.4)$$

3.2.1.3 Allocation Algorithms

If a request can be allocated to multiple resources, the broker has to determine which resource is best suited to serve the request. The broker can use either First-Fit (FF) or Next-Fit (NF) algorithm [33] to perform the allocation of the request.

- *First-Fit (FF) algorithm*: In this algorithm, the broker iterates the list of available resources starting from the beginning of the list and chooses the first resource that can serve the request.
- *Next-Fit (NF) algorithm*: In this algorithm, the broker keeps track of the resource that was last assigned a request. The broker iterates the resource list starting with a resource that appears directly after the aforementioned resource and chooses the first available resource that can serve the request.

3.2.2 Handling Underestimation of Request Runtimes for Executing Requests

This section addresses scenarios in which request runtime is underestimated. Underestimation of request runtime is caused by either the system or the user.

If the request runtime is found to be underestimated by the user, it is aborted by the system. If the system underestimates the request runtime, the request is pre-empted and allowed to run later — when there is a free slot in the resource schedule. If the request does not complete before the end of this time slot, it is preempted again and assigned the next free time slot in the resource schedule. This continues until the request is completed or reaches its estimated runtime provided by the user at which it is aborted.

3.3 Algorithm 2 - A Variation of Algorithm 1

Algorithm 2 is similar to Algorithm 1 except in the way it handles requests with underestimated runtimes. With Algorithm 2, requests with underestimated runtimes performed by users or the system are not aborted. Instead, when a request runs up to its allocated time and has not finished, it is pre-empted and allowed to run later when there is a free time slot in the resource schedule. If the request does not complete before the end of this time slot, it is preempted again and assigned the next free time slot in the resource schedule. This continues until the request is completed. The request is allowed to run to completion regardless of whether it misses its deadline or not.

4 Chapter: Simulation Model

A simulator was developed for the research presented in this thesis. This chapter describes the simulator and the experiments that have been used to study the performance of the matchmaking strategies.

4.1 Performance Metrics

This section describes the performance metrics that are used to compare the performance of different algorithms in the experiments.

4.1.1 Probability of Blocking (P_b)

The probability of blocking is the probability that the system will reject a request [13], [14]. P_b is calculated as the ratio of total number of rejected requests over the total number of requests arriving at the system:

$$P_b = \frac{\sum \text{Number of rejected requests}}{\sum \text{Number of submitted requests}}$$

4.1.2 Rate of Accepted requests (A)

The rate of accepted requests denotes the number of requests accepted by the system per unit time [31], [33]. It is calculated as the product of the acceptance ratio with the arrival rate λ , described in Section 0. The rate of accepted requests captures the effect of arrival rate that is absent in P_b . Acceptance ratio is the ratio of the number of accepted requests over the total number of requests submitted to the system and it is given by $(1 - P_b)$. The formula for calculating the rate of accepted requests is shown below:

$$A = (1 - P_b) \times \lambda$$

4.1.3 Percentage of Work Rejected (W_R)

Percentage of work rejected is the percentage of the overall work rejected by the system [13], [14]. W_R is calculated as the ratio of the total work of all the requests rejected by the system over the total work of all the requests submitted to the system. The formula for calculating the percentage of work rejected is shown below:

$$W_R = \frac{\sum \text{Runtimes of all requests rejected}}{\sum \text{Runtime of all requests submitted}} \times 100$$

Although W_R is related to the requests rejected, it contains additional information regarding the runtimes of requests rejected that is not captured in P_b or A .

4.1.4 Resource Utilization (U)

Resource utilization is the fraction of time the resource is busy executing requests. It is calculated as the ratio of total busy time of the resource over the total simulation duration. The resource utilization is calculated as the following:

$$U = \frac{\sum \text{busy time}}{\text{simulation duration}}$$

4.1.5 Useful utilization (UU)

Useful utilization for a resource is the fraction of time the resource is busy executing requests that complete successfully. It is an important metric for determining the efficiency of the system when there are runtime estimation errors in the requests [13], [14]. It is calculated as the following:

$$UU = \frac{\sum \text{resource busy time serving completed requests}}{\text{simulation duration}}$$

Although all the three performance metrics, P_b , A , and W_R , are related to requests rejected by the system, each provides different information to the readers. P_b is the proportion of rejected requests on the system. By capturing the impact of λ , A indicates the potential revenue earned per unit time by the service provider at a given arrival rate. W_R includes the effect of execution times of requests that is not captured by A or P_b . It also indicates potential revenue lost by not accepting the rejected requests.

4.2 GridSim Toolkit

The GridSim Toolkit is a widely-used simulation tool for simulating grid and cloud systems [32]. It allows users to build simulation models for grids and clouds with various components. As described in the GridSim website [20], it provides a comprehensive facility for simulation of different classes of heterogeneous resources, users, applications, resource brokers, and schedulers.

The toolkit allows users to configure the characteristics of the resources and brokers to suit their own needs. The configurable characteristics include types of operating systems, number of processors, and the cost of using the resource. Moreover, there are some standard scheduling policies for the resources provided in GridSim. However, it allows users to provide their own implementation of new scheduling policies.

GridSim is built on top of existing software technologies following a modular and multi-layered architecture [3]. There are five layers, as illustrated in Figure 4.1. The first layer comprises the Java Virtual Machine (JVM). JVM provides an environment where applications written in Java can be executed. The second layer utilizes a modified version of SimJava2 [20], which is a discrete-event simulation package written for Java. The

third layer is responsible for performing simulation of core grid entities, such as resources and information services. Simulation of GridSim entities such as resource brokers and resource schedulers is managed by the fourth layer. The fifth layer provides services for users to devise scenarios to evaluate the performance of resource management strategies, scheduling policies, and other types of algorithms.

Figure 4.1: GridSim architecture adopted from [43]

4.3 Simulation Entities

This section describes all the entities involved in the simulation model.

4.3.1 Source

This is the component that is responsible for sending requests to the broker, which is described in Section 4.3.2, and processing the statistical data. The required number of requests along with the desired characteristics is generated by this source. The simulation starts when the source sends the first request and ends when the last request finishes executing.

4.3.2 Broker

The broker is the component that is responsible for matchmaking requests to appropriate resources. Before any allocation can be performed, the broker retrieves a list of available resources in the grid from the Grid Information Service. After the list of resources is obtained, the broker uses an allocation algorithm to perform matchmaking for the incoming requests.

4.3.3 Grid Information Service (GIS)

Grid Information Service (GIS) is used for resource management in grids and clouds. It maintains a list of all the available resources in the system. When a resource is available for use, it has to notify the GIS for its availability. For resource discovery, the broker interacts with GIS in order to get the list of available resources in the system.

4.3.4 Resource

This entity simulates a computing resource in grids and clouds. The GridSim toolkit provides two kinds of resources. The first type of resource can only handle OD requests.

The second type of resource can handle both AR and OD requests. The second type of resource is in the simulation.

4.3.5 Resource Characteristics

Every resource contains a resource characteristics entity. This entity stores all information about the resource, such as the number of CPUs in the resource and the operating system. In the simulator program implemented in this research, the resource characteristic entity is extended to keep track of the number of requests currently scheduled at the resource. The broker can retrieve this information in order to perform matchmaking for requests. In this thesis, every resource is assumed to have a single CPU while other values are kept unchanged from the values provided by GridSim (see Table 1).

Table 1: Default values of resource characteristics provided by GridSim

Fields	Values
Time zone	9.0 (GMT+9)
MIPS rating	377
Architecture	Sun Ultra
Operating system	Solaris
Peak load	0
Off-peak load	0

4.3.6 Resource Scheduling Policy

The scheduling policy determines how an incoming request is scheduled on the resource. When a request arrives at the resource, the resource tries to schedule the request using the chosen scheduling policy. If the schedule allows the request to meet its deadline and does not cause other requests in the resource to miss their deadlines, the request is accepted and scheduled on the resource. If either of the aforementioned requirements is not met, the request is rejected.

In the thesis, Earliest Deadline First (EDF), Minimum Laxity First (MLF), and Earliest Start Time First (ESF) scheduling policies that follow work-conserving principle are used. Furthermore, the broker is assumed to be not aware of the scheduling policy of the resource and the AS criterion is thus used. The scheduling policies component of the simulator was adapted from [31]. Appropriate modifications of the code from [31] were performed to adapt the component to the other simulator components developed by me for this research.

4.3.6.1 Earliest Deadline First Scheduling Policy

In Earliest Deadline First (EDF) scheduling policy [31], [32], requests are ordered based on their deadlines. A request having an earlier deadline is allowed to execute sooner than requests having later deadlines. If there is a free time slot in the resource schedule, a request ready for executing at a time included in the free time slot may be allowed to start even if its deadline is not the earliest among the requests on the resource. This ensures that the work-conserving principle is maintained.

4.3.6.2 Earliest Start Time First Scheduling Policy

In Earliest Start time First (ESF) scheduling policy [31], requests on a resource are ordered based on their earliest start times. A request having a smaller earliest start time is allowed to execute sooner than requests having higher earliest start times.

4.3.6.3 Minimum Laxity First Scheduling Policy

In Minimum Laxity First (MLF) scheduling policy [13], requests are ordered based on their laxities (see Section 1.1.5.2). Requests having smaller laxities are allowed to execute sooner than requests having larger laxities. If there is a free time slot in the resource schedule, a request ready for executing at a time included in the free time slot may be allowed to start even if its laxity is not the smallest among the requests on the resource. This ensures that the work-conserving principle is maintained.

4.3.7 Request (Gridlet)

This entity contains information about a request that will be running on a resource. This entity is referred to as a gridlet in the GridSim toolkit. A gridlet contains information about the desired number of CPUs, the type of operating system, the user estimation of the request runtime, and so on. However, only the following fields used in the entity are controlled in this research:

- Earliest start time of the request
- Deadline of the request
- User estimated runtime for the request
- Actual runtime the request needs

- Arrival time of the request

Section 4.4.8 has a detailed description on how the values of the aforementioned fields are generated. The other fields are kept unchanged using the default values provided by GridSim [20] (see Table 2).

Table 2: Default values of gridlet characteristics provided by GridSim

Fields	Values
Gridlet length	400 MI (million instructions)
Gridlet file size	400 bytes
Gridlet output size	400 bytes

4.4 System and Workload Parameters

This section describes the parameters of the simulation that are of interest. First, the parameters of the system are described (from Section 4.4.1 to 4.4.8). Then Section 4.4.8 describes the parameters that define the workload the system is subjected to.

A factor-at-a-time approach is used in the simulation. One parameter is varied in an experiment while other parameters are held at their default values. The values of each parameter used with its default value are shown in Table 3 and Table 4.

4.4.1 Proportion of Allowable Missed Deadlines (p_{miss})

This refers to the proportion of requests with missed deadlines that a user is willing to accept. This value is used by the matchmaking algorithms to determine whether to compress the estimated runtime of requests or not as described in Section 3.2. In order to

determine the impact of p_{miss} on the performance of the algorithms, the thesis experiments with different values of p_{miss} (see Table 3).

Table 3: System parameters, algorithms, and scheduling policies

Parameter	Key parameter/ Algorithms/ Scheduling Policies	Key parameter value	
		Default value	Parameter values
Number of resources (R)	R	3	1, 3, 6
Allocation algorithms	A_p	FF	FF, NF
Resource scheduling policy	Sched_Policy	EDF	EDF, ESF, MLF
Proportion of allowable missed deadline for the users	p_{miss}	10%	5%, 7.5%, 10%, 12.5%, 15%
Margin	Δ	5%	2%, 4%, 5%, 7%, 9%
The window size	W	10	2, 5, 10, 20, 40

4.4.2 Margin (Δ)

This refers to the margin value Δ used in Inequality 3.1 for determining when matchmaker should use the system estimate of request runtime for matchmaking as described in Section 3.2. The thesis experiments with different values of Δ to determine its impact on the performance of the matchmaking algorithms (See Table 3).

4.4.3 Window Size (W)

A sliding window of size W contains requests that are used for calculating the average percentage of overestimation. The average percentage of overestimation is used for calculating the system estimate of request runtime used in the matchmaking algorithms described in Section 3.2.1.1. The values of W along with its default value used in the thesis are reported in Table 3.

4.4.4 Number of Resources (R)

This refers to the number of available resources in the system for allocating requests to. This thesis experiments with different values of R to study its impact on the performance of the matchmaking algorithms. The values of R along with the default value are similar to the values used in [13], [31], [42] are shown in Table 3.

4.4.5 Allocation Algorithm (A_p)

The broker uses the allocation algorithm for allocating requests to the appropriate resources. In the simulation, the First-Fit algorithm is used as the default allocation algorithm for the broker. In order to study the impact of the allocation algorithms, Next-Fit algorithm is used and compared with the performance of First-Fit algorithm. The descriptions of these two algorithms are provided in Section 3.2.1.3. The values of A_p along with the default value are similar to the ones used in [29] and are reported in Table 3.

4.4.6 Resource Scheduling Policy (Sched_Policy)

This is the scheduling policy that is used at the resource. In order to study the impact of scheduling policy on performance, the thesis uses EDF, ESF, and MLF scheduling policies with EDF as the default scheduling policy. Similar scheduling policies were used in [13], [31]. The descriptions of these algorithms are provided in Section 4.3.6. The values of Sched_Policy along with the default value are reported in Table 3.

4.4.7 Types of Workload

This refers to the percentage of on-demand and advance reservation requests in the workload sent to the system. As this thesis focuses solely on the AS criterion, AR requests are the only type of requests that are used as the workload.

4.4.8 Workload Model

The thesis uses an open model where a stream of requests arrives at the system. As specified in Section 4.4.8, this research considers systems subjected to Advance Reservation requests. The details of the workload used are described in the following sections.

4.4.8.1 Request Arrival Rate

The arrival rate of the requests, denoted by λ , defines the rate at which the requests arrive at the system. It is measured by the number of requests arriving per second at the system. The request arrivals can be adequately modeled by using a Poisson process. The rationale for choosing Poisson process was given in [21]. In addition, many researchers such as [7], [21], [33] have used a Poisson process for modeling request arrivals in the context of

grids and clouds. In line with other researchers, the thesis uses Poisson process for modeling request arrivals. The values of λ are shown in Table 4.

Table 4: Summary of the workload parameters

Parameter	Distribution	Key parameter	Key parameter value	
			Default value	Parameter values
Arrival process	Poisson	λ	0.005333333 requests/sec	0.0009, 0.0015, 0.003, 0.007, 0.0085, 0.01, 0.011 requests/sec
Percentage laxity	Uniform	L (min = 0%, max = 2×L)	400%	50%, 100%, 200%, 400%, 700%
Time between the arrival of an AR and its earliest start time	Uniform	T_s (min = 0 hour, max = T_s)	12 hours	5, 12, 18, 24 hours
Request runtime	Uniform	S	50 minutes	10 minutes [min = 2 minutes, max = 18 minutes], 50 minutes [min = 10 minutes, max = 90 minutes], 150 minutes [min = 30 minutes, max = 270 minutes]
	Hyper-exponential	CoV, S	CoV = 1 S = 50 minutes	CoV = 1, 2, 3

4.4.8.2 Request Runtime

In order to model the runtime of the requests, a uniform distribution with a minimum value of 10 minutes and maximum value of 90 minutes is used. Thus, the mean runtime of the requests, denoted as S , is 50 minutes. A similar distribution is used by other researchers (see [32] for example).

To study the impact of high variability in request runtimes, the thesis uses hyper-exponential distribution for request runtimes. The mean request runtimes is kept at 3000 seconds, which is the default mean request runtime of 50 minutes used in other experiments, while coefficient of variation (CoV) is varied. The values of CoV are similar to [31] and are reported in Table 4.

4.4.9 Model of Error in User Estimation of Request Runtime

In order to model the error in user estimation of request runtimes, this thesis uses the model described in [13], [14] which is derived from traces collected from IBM SP2 supercomputer as the default model for error in user estimation of request runtime. This model is described in Section 4.4.9.1.

In order to study the effects of error in user estimation of request runtime on the performance of the SAR based matchmaking algorithms, this thesis uses two additional error models, which are described in Section 4.4.9.2 and Section 4.4.9.3. Parameter values for all these error models are shown in Table 5.

Table 5: Summary of models of errors associated with user estimation of request runtimes

Error in user estimate of request runtimes	Distribution	Key parameter	
Default model	Hyper-exponential, Exponential, and Uniform	See Section 4.4.9.1	Based on SP2 logs (see section 4.4.9.1)
Constant error model	Fixed	Percentage of overestimated	50%
		Percentage of underestimated	50%
Feitelson's error model	Uniform	See Section 4.4.9.3	---

4.4.9.1 Default Model of Error in User Estimation of Request Runtime

In this model, the error in user estimation of request runtime is quantified by the percentage error in user estimation of request runtime. The percentage error in user estimation of request runtime, called η , is defined as the following:

$$\eta = \frac{\text{Estimated runtime} - \text{actual runtime}}{\text{actual runtime}} \times 100$$

From the equation above, a positive value of η indicates that the runtime of the request is overestimated. In contrast, a negative value of η indicates underestimation.

In this case, the error in user estimation of request runtime is modeled based on the analysis of the traces collected from IBM SP2 supercomputer that were made

available in [15]. In their analysis in [13], [14], the authors showed that almost 98% of the requests have their runtime overestimated. In some of the requests, the request runtimes are overestimated by as much as 25,000%. In addition, the analysis shows that almost 2% of the requests have underestimated runtimes.

In order to model η , [13], [14] models η for requests with overestimated and underestimated runtimes separately.

In the thesis the number of times the request runtimes are underestimated by the users is 2%. η is modeled using a uniform distribution with minimum value of η at -82% and maximum value of η at 0% [13], [14].

The percentage of requests with overestimated runtimes is 98% of the total generated requests. For these requests, the requests are divided into different categories based on the value of their actual runtimes. η in each category is modeled by a hyper-exponential distribution with the rate parameters θ_1 (for the first exponential distribution), θ_2 (for the second exponential distribution), and p_h (which is the probability of the first distribution is used). For requests having logarithm of runtimes from 4 to 5, η can be modeled using exponential distribution with rate parameter θ [13], [14]. Additionally, in each category, there are a certain number of requests that are terminated abnormally. As a result, η of these requests are really large. η for these requests is modeled using a uniform distribution where L and H are the minimum and maximum values of the uniform distribution. The values of the parameters of these distributions are adopted from [13], [14] and shown in Table 6.

4.4.9.2 Model with Constant Error in User Estimation of Request Runtimes

In this model, all request runtimes are either overestimated or underestimated by a constant percentage of their actual runtimes. This model is used in [12]. This thesis studies the effect of requests that are either overestimated or underestimated by 50% of their actual runtimes, which is the middle values of the range of values used in [12].

Table 6: Parameter values for η

Range of request runtimes (seconds)	Distribution (D) for η	Parameters for distribution D	Probability of abnormal termination	Parameters for uniform distribution
1 – 100	Hyper-exponential	$\theta_1 = 0.00082$ $\theta_2 = 0.00025$ $p_h = 0.9$	0.21	$L_1 = 10,000\%$ $H_1 = 25,000\%$
100 – 1,000	Hyper-exponential	$\theta_1 = 0.00128$ $\theta_2 = 0.00024$ $p_h = 0.9$	0.09	$L_1 = 10,000\%$ $H_1 = 20,000\%$
1,000 – 10,000	Hyper-exponential	$\theta_1 = 0.00300$ $\theta_2 = 0.00137$ $p_h = 0.9$	0.03	$L_1 = 3,000\%$ $H_1 = 10,000\%$
10,000 – 100,000	Exponential	$\theta = 0.00550$	0.01	$L_1 = 400\%$ $H_1 = 600\%$

4.4.9.3 Feitelson's Model of Error in User Estimation of Request Runtimes

This model is introduced by Feitelson and Mu'alem in [35]. The model was based on the analysis performed by the authors of real workload traces collected from Cornell Theory Center (CTC), Swedish Royal Institute of Technology (KTH), and San Diego Supercomputer Center (SDSC). The user estimated runtime of a request, denoted as T_E , is modeled as follows:

1. With a probability of 10%, $T_E = 0.99 \times T_R$, where T_R is the actual runtime of the request
2. With a probability of 90%, T_E is modeled based on the value of T_R
 - a. If $T_R < 90$ seconds, $T_E = 10 \times T_R$
 - b. Otherwise, $T_E = \frac{T_R}{u}$ where u is uniformly distributed in the range $(0, 1]$

If the value of T_E is very large, it is truncated to a predefined value.

In this thesis, a predefined value is 27 hours which is higher than the largest possible value that T_E can attain is used.

4.4.10 Additional Workload Parameters

4.4.10.1 Earliest Start Time of Request

The earliest start time of a request is the sum of its arrival time and a uniformly distributed number in the range $(0 \text{ hour}, T_S \text{ hours})$. The value of T_S is chosen to be 12 hours as the default value for all the experiments. This value is used by other researchers as well (see [13], [14], [31], [32], [40] for example). In order to study the effect of T_S on the performance of the algorithms, different values of T_S are used in the thesis. The values of T_S are reported in Table 4.

4.4.10.2 Request Laxity

The percentage laxity of an AR request is defined as the ratio of the request's laxity, described in Section 1.1.5.2, and its actual runtime multiplied by 100 [13], [14]. It is modeled using a uniform distribution with minimum value of 0%. The mean of the uniform distribution, denoted as L , is varied between 0% and 700%. The default value of

L is 400% which is similar to the value used in [13], [14]. The values of L are reported in Table 4.

4.5 Accuracy of Results

The simulation is run long enough to allow the system to reach a steady state. Repeating the simulation experiments multiple times allows the computation of confidence intervals. Each simulation is run for 80,000 request arrivals and repeated multiple times such that a confidence interval lower than or equal to $\pm 5\%$ for P_b , A, W_R , U, and UU are achieved at a confidence level of 95%.

5 Chapter: Performance Evaluation

Simulations were conducted to analyze the performance of the algorithms, Algorithm 1 and Algorithm 2, under different workload and system parameters. The two algorithms are described in greater detail Chapter 3.

In addition, the performance of the algorithms was compared with two different cases. In the first case, abbreviated as NE (No-Error), there was no error in the user-estimated runtime. In the second case, abbreviated as NM (No-Modification), there was error in user-estimated runtime; however, the broker accepted the requests without making any modifications to the estimated runtime. In both cases, the AS criterion is used for matchmaking requests.

As indicated in Section 4.5, the simulation experiments are run long enough and repeated multiple times so that a confidence interval that is lower than or equal to $\pm 5\%$ at a level of 95% is achieved.

5.1 Impact of Arrival Rate (λ)

This section presents the impact of arrival rate on the performance of the algorithms. In these experiments, the arrival rate (λ) is varied in order to subject the system to different loads, starting from low and moving to high.

As λ increases, the system is subjected to a greater amount of work. Since the system has a limited capacity, more work causes it to reject more requests. Figure 5.1 confirms that the P_b values for all four algorithms increased as λ increased. In Figure 5.1, Algorithm 1 and Algorithm 2 achieved lower P_b values than NM for low and medium

arrival rate because they compressed the user estimated runtimes of requests, whereas NM did not. Since request runtimes are typically overestimated by users [1], [13], [14], compressing the estimated runtime allows more requests to be accepted and thus results in lower P_b for Algorithm 1 and Algorithm 2.

Because NE knows the exact runtime of the requests, it produces the lowest P_b values of the four algorithms at low and medium λ , as shown in Figure 5.1. At high λ , P_b values achieved with all these algorithms are statistically comparable to one another.

Figure 5.1: Impact of arrival rate on P_b

In Figure 5.2, all algorithms have increased values of A as λ increases because the system is subjected to a higher rate of incoming requests. Although P_b increases with λ , there is a greater number of the requests accepted per second. As more requests are accepted on the system, a higher potential revenue can be achieved for the service provider. Although more requests are rejected at higher λ , A is observed to improve with an increase in λ . The increase in A with λ captured in Figure 5.2 shows the potential

increase in revenue for high value of λ . In Figure 5.2, NE has the highest A values because it rejects the lowest number of requests. NM has the lowest A values because it rejects the highest number of requests. Algorithm 1 and Algorithm 2 compress the estimated runtime of the requests and do not reject many requests unnecessarily. Therefore, they have higher A values than NM even though they demonstrate a comparable performance with NM in terms of P_b at higher values of λ .

Figure 5.2: Impact of arrival rate on A

In Figure 5.3, all algorithms exhibit an increase W_R as λ increases. Since the system has a limited capacity, subjecting it to more work by increasing λ values increases the chance that it will reject work.

Algorithm 1 and Algorithm 2 achieve lower W_R than the NM when λ increases because they reject a lower number of requests compared to NM for low and medium λ , as shown in Figure 5.1. NE achieves the lowest W_R for low and medium λ due to its

rejecting the least number of requests, as shown in Figure 5.1. At high values of λ , all these algorithms achieve statistically similar performance in terms of W_R .

Figure 5.3: Impact of arrival rate on W_R

Figure 5.4 shows that U for all of the algorithms increases as λ increases. Since increasing λ allows the system to accept more requests, there is a higher probability of fitting the requests to the resources, which drives up the utilization of the resources. As shown in Figure 5.4, NE has the highest U because it correctly estimates request runtimes, thus allowing the resources to be highly utilized. Algorithm 1 and Algorithm 2 achieve a higher U value than NM.

Even though both Algorithm 1 and Algorithm 2 have similar W_R , Algorithm 1 has a lower U than Algorithm 2. This is because Algorithm 1 aborts requests whose runtimes are underestimated by users. Since it does not allow these requests to run to completion, their time spent at the resources must be less than the time they would have spent at the resources had they not been aborted. This reduces the U values of the resources.

Figure 5.4: Impact of arrival rate on U

Figure 5.5 shows the UU for different values of λ . In the graph, UU increases as λ increases. This is because U for all algorithms increases as λ increases.

NM demonstrates the worst performance. The graph shows that Algorithm 2 and Algorithm 1 achieve a significant higher UU than NM at the cost of only 10% requests missing deadlines. It shows that allowing a small proportion of requests to miss deadlines improve UU achieved with Algorithm 1 by more than 65% as compared to UU achieved with NM. UU achieved with Algorithm 2 increases by almost 100% as compared to UU achieved with NM. Algorithm 2 and NE achieve a higher UU in comparison to Algorithm 1 because unlike Algorithm 1, they do not abort any requests. Algorithm 1 aborts requests with underestimated runtimes caused by the users. Since the time serving these aborted requests does not get counted towards UU, it affects UU achieved by Algorithm 1. In addition, the graph shows that Algorithm 2 achieves a higher UU in comparison with Algorithm 1 at higher λ . This indicates that allowing requests with

underestimated runtimes caused by users to run to completion improves the performance of the system.

Figure 5.5: Impact of arrival rate on UU

5.2 Impact of Time between the Arrival Time and the Earliest Start Time of AR Request

In all the experiments, the earliest start time of a request is generated by taking a sum of the request's arrival time and a value generated from a uniform distribution with a minimum value of 0 hours and a maximum value of T_S (which is equal to 12 hours by default). To study the effect of the earliest start time on the performance of the algorithms, this set of experiments studies varied values of T_S : 1 hour, 5 hours, 18 hours, and 24 hours.

Figure 5.6 shows the impact of T_S on P_b values for all algorithms. As T_S increases, the P_b values for both NE and NM slightly increase. This is because increased T_S results in requests staying at the resource longer, thus increasing the chance that these requests

will overlap with incoming requests. This leads to a greater rejection of incoming requests.

Figure 5.6: Impact of T_s on P_b

The P_b values of Algorithm 1 and Algorithm 2 demonstrate a non-monotonic behavior. P_b tends to decrease slightly as T_s approaches 12 hours. Low T_s (lower than 12 hours) causes requests to start sooner and thus increases contention at the resources. An increase in T_s (up to 12 hours) spreads out the start time of requests and allows more flexibility for the scheduler to order the requests. Thus, it allows more requests to be accepted. Increases in T_s beyond 12 hours cause requests to stay at the resource longer, and increases the chance of overlap with incoming requests. This is why the P_b of these algorithms tends to increase slightly with an increase in T_s .

Algorithm 1 and Algorithm 2 have slightly lower P_b values than NM as discussed earlier. This is because Algorithm 1 and Algorithm 2 compress estimated runtimes of requests and avoid many unnecessary rejections due to user overestimation, as is the case

with NM. Because NE does not have errors in estimated runtime, it has the lowest values of P_b of all four algorithms (see Figure 5.6).

W_R shows similar behavior as P_b when T_S increases. As A is calculated as $(1 - P_b)\lambda$, the graph of A values also show a similar behavior as P_b when T_S increases (See Figure A-1 and Figure A-2 in Appendix A.1).

Figure 5.7 shows the impact of T_S on U for all algorithms. Both NM and NE produce increased P_b values as T_S increases. This leads their P_b values to increase accordingly. As the amount of work accepted into the system decreases, the resources become more idle, and the value of U for these algorithms decreases.

Algorithm 1 and Algorithm 2 have slightly reduced P_b values as T_S increases up to 12 hours. As the number of requests accepted into the system increases, resource utilization increases. Thus, these algorithms have increased U values. When T_S is greater than 12 hours, the P_b of Algorithm 1 and Algorithm 2 increase as T_S increases. Reducing the number of requests accepted into the system means that there is not much work for the resources. Therefore, the resources become more idle, causing these algorithms to have reduced U values.

In addition, Algorithm 1 aborts requests with underestimated runtimes caused by users, preventing these requests to run to completion. Meanwhile Algorithm 2 does allow requests with underestimated runtimes to run to completion. This is why Algorithm 1 has lower U than Algorithm 2 even though they accept a comparable number of requests.

As UU is only less than U if the system has to abort requests, only Algorithm 1 and NM have UU lower than U . Algorithm 2 and NE have their UU values equal U (See Figure A-3 in Appendix A.1).

Figure 5.7: Impact T_s on U

5.3 Impact of Laxity

This section studies the impact of laxity on the performance of the algorithms. In this set of experiments, the mean percentage of laxity (L), described in Section 4.4.10.2, is varied to subject the system to different workloads.

Figure 5.8 shows the impact of L on the values of P_b achieved by all algorithms. As L increases, P_b for all algorithms slightly decreases. With a higher L , requests have more laxity and thus can give the matchmaker more flexibility in matchmaking requests. As a result, there is a greater chance of the requests being accepted into the system. In Figure 5.8, P_b values achieved with all algorithms are statistically comparable to one another.

Figure 5.8: Impact of L on P_b

As A is calculated as $(1 - P_b)\lambda$, A can be derived from P_b (See Figure A-4 in Appendix A.2). W_R shows a similar behavior as P_b when L increases (See Figure A-5 in Appendix A.2).

Figure 5.9 shows that all algorithms achieve higher U as L increases. A small increase in the number of accepted requests, as shown in Figure 5.8, allows resources to be utilized more and increases U . NE accepts the highest number of requests and achieves the highest U . NM achieves the lowest U . Algorithm 1 and Algorithm 2 display higher U than NM because they accept more requests than NM.

As discussed in the previous cases, UU is less than U if the system has to abort requests; only Algorithm 1 and NM have UU values lower than their U values. Algorithm 2 and NE have their UU values equal to U (See Figure A-6 in Appendix 5.3).

Figure 5.9: Impact of L on U

5.4 Impact of the Window Size

This set of experiments studies the impact of window size (W) on the performance of the algorithms. As NE and NM do not use W , changes in the value of W do not have any impact on these algorithms.

Figure 5.10 shows that Algorithm 1 and Algorithm 2 have unchanged values of P_b when W is greater than 10. When W is less than 10, the P_b achieved with Algorithm 1 and Algorithm 2 slightly decreases as W increases. A low W value means that there a small number of requests is used in the calculation of estimated runtime of an incoming request. As a result, there is a higher chance of the matchmaker making mistakes in predicting request runtime, causing more requests to miss their deadlines. As the proportion of missed deadline requests increases near the value of p_{miss} , the matchmaker has to stop compressing requests' estimated runtimes. It then has to use the estimated

runtimes provided by users. Since users often overestimate request runtimes, the matchmaker rejects more requests and thus results in higher values of P_b .

As the computation of A is based on P_b , the behavior A also shows a similar trend (See Figure A-7 in Appendix A.3). W_R shows a similar behavior as P_b when W is varied (See Figure A-8 in Appendix A.3).

Figure 5.11 shows that U achieved with Algorithm 1 and Algorithm 2 remains unchanged when W is greater than 10. As P_b remains unchanged, the U values achieved by these algorithms remain unchanged (See Figure 5.11). For smaller value of W ($W < 10$), U is observed to increase with W because of the small increase in the number of requests accepted at the higher value of W .

Figure 5.10: Impact of W on P_b

Algorithm 1 aborts requests with underestimated runtimes caused by users while Algorithm 2 allows all requests to run to completion. As the time served for an aborted request is always shorter than the actual runtime of the request, aborting the request

reduces the time of the resource serving that request. Therefore, the value of U achieved by Algorithm 1 is lower than that achieved by Algorithm 2.

Figure 5.11: Impact of W on U

As UU is only less than U if the system aborts requests, only Algorithm 1 and NM have UU lower than U . Algorithm 2 and NE have their UU values equal U (See Figure A-9 in Appendix A.3).

5.5 Impact of Margin (Δ)

This section presents the impact of Δ on the performance of all the algorithms. As NE and NM do not use this value, changes in Δ do not have any impact on the performance of either NE or NM.

Figure 5.12 that when the value of Δ increases beyond 5%, the P_b achieved with Algorithm 1 and Algorithm 2 achieves a small increase in its value. This is because Δ determines how close the proportion of requests that miss their deadlines is to the value

of p_{miss} . A high Δ reduces the actual proportion of requests with missed deadlines that Algorithm 1 and Algorithm 2 achieve. Therefore, for high value of Δ , Algorithm 1 and Algorithm 2 cannot compress the user estimated runtimes of as many requests as they can when Δ is low. A greater number of requests with uncompressed runtimes increase their chances of being rejected because most of the requests' runtimes are highly overestimated. As a result, P_b increases with an increase in Δ .

Figure 5.12: Impact of Δ on P_b

When Δ is less than 5%, the P_b values of Algorithm 1 and Algorithm 2 remain unchanged with an increase in Δ . For reasons discussed earlier, W_R and A show a similar behavior as P_b when Δ varies (See Figure A-10 and Figure A-11 in Appendix A.4).

Figure 5.13 shows that, for Δ greater than 5%, the U values of both Algorithm 1 and Algorithm 2 become smaller as Δ increases. This is because the P_b value for these algorithms increases slightly when Δ is increased. As the amount of requests accepted

into the system decreases, the U values of the resources achieved with Algorithm 1 and Algorithm 2 decrease.

For Δ less than 5%, U achieved with Algorithm 1 and Algorithm 2 remain unchanged because the amount of work rejected remains unchanged.

For reasons discussed earlier, UU is less than U if the system aborts requests. Thus, only Algorithm 1 and NM have UU values lower than U values whereas Algorithm 2 and NE have their UU equal to U (See Figure A-12 in Appendix A.4).

Figure 5.13: Impact of Δ on U

5.6 Impact of the Proportion of Allowable Missed Deadlines (p_{miss})

This section presents the impact of p_{miss} on the performance of all the algorithms. As NE and NM do not use p_{miss} in their algorithms, changes in the values of p_{miss} do not have any impact on the performance of these algorithms.

Figure 5.14 shows the effect of p_{miss} on P_b for Algorithm 1 and Algorithm 2. The graph shows that when p_{miss} is less than 10%, an increase in the value of p_{miss} decreases P_b achieved with Algorithm 1 and Algorithm 2. Since p_{miss} indicates the proportion of requests that are allowed to miss their deadlines, a higher value of p_{miss} allows the SAR based matchmaking algorithms to compress more requests, helping to reduce the number of requests whose runtimes are overestimated by users. In turn, this increases the system's chances of accepting more requests. It also explains why P_b for Algorithm 1 and Algorithm 2 tends to decrease slightly for increases in p_{miss} up to 10%.

Figure 5.14: Impact of p_{miss} on P_b

The graphs for Algorithm 1 and Algorithm 2 flatten off when a value of approximately 10% is reached by p_{miss} .

Once again, W_R and A show similar behaviors as P_b (See Figure A-13 and Figure A-14 in Appendix A.5).

Figure 5.15 shows the effect of p_{miss} on U . When p_{miss} is less than 10%, the U values of Algorithm 1 and Algorithm 2 increase as p_{miss} increases. This is a direct effect of a decrease in P_b with an increase in p_{miss} (See Figure 5.15). As a result, the resources are utilized more since they are given more work to do. When p_{miss} is greater than 10%, increases in p_{miss} do not cause an increase in U values for Algorithm 1 and Algorithm 2. As shown in Figure 5.15, P_b remains unchanged when p_{miss} is greater than 10%. As expected, U achieved by Algorithm 1 and Algorithm 2 also remains unchanged with an increase in p_{miss} .

Figure 5.15: Impact of p_{miss} on U

Because of the relationship between U and UU discussed earlier, only Algorithm 1 and NM display UU values lower than their respective U values. Algorithm 2 and NE have their UU values equal to their respective U (See Figure A-15 in Appendix A.5).

5.7 Impact of Number of Resources (R)

This set of experiments studies the effect of R on the performance of the algorithms. In the experiments, R is given three values: 1 resource, 3 resources, and 6 resources.

Figure 5.16 shows the impact of R on the P_b for all algorithms. The graph shows that the P_b values of NE and NM decrease slightly as R increases. A higher decrease in P_b is expected at higher arrival rates. As more resources are added to the system, the system is better able to serve more requests. This increases the number of requests that can be accepted into the system.

Figure 5.16: Impact of R on P_b

The P_b values of Algorithm 1 and Algorithm 2 decrease up to an R value of 3 resources. When R is greater than 3, the P_b achieved by these algorithms flattens off as R increases. This behaviour is the result of a greater number of requests with underestimated runtimes caused by the matchmaker being accepted with the addition of more resources. These requests are more likely to miss their deadlines, thus increasing

the proportion of requests with missed deadlines. In turn, the algorithms stop compressing the user estimation of the request runtimes more often to avoid causing more requests with missed deadlines and violating the limit on the proportion of requests with missed deadlines allowed. As the number of requests with uncompressed estimated runtimes increases, this reduces the number of requests accepted to the system as compared to cases with small R (R is less than 6). This explains why the P_b values of Algorithm 1 and Algorithm 2 flatten off when R exceeds 3.

Of the four algorithms, NE has the lowest P_b due to having no error in estimation for any given R . NM has the highest P_b because it rejects many requests with highly overestimated runtimes unnecessarily. Algorithm 1 and Algorithm 2 have lower P_b values than NM because they compress the estimated runtime of the requests and avoid unnecessary rejections of requests.

For reasons discussed earlier, W_R and A show similar behaviors as P_b (See Figure A-16 and Figure A-17 in Appendix A.6).

Figure 5.17 shows the effects of R on U for all the algorithms. It shows that NE and NM achieve an increase in the values of U when R increases. This is because more requests are accepted, allowing resources to be utilized more. Algorithm 1 and Algorithm 2 have decreased P_b values when R increases up to 3. As the number of requests accepted into the system increases, resources utilization increases, causing an increase in the values of U . However, both algorithms produce an increased P_b when R increases beyond 3 resources. As the number of requests accepted to the system decreases, resource utilization decreases.

Of the four algorithms, NE has the highest U and NM has the lowest U . Algorithm 1 and Algorithm 2 display intermediate values of U . Algorithm 1 aborts requests with underestimated runtimes caused by the users whereas Algorithm 2 allows these requests to run to completion. As the time served for an aborted request is always shorter than the actual runtime of the request, aborting the request reduces the time of the resource serving that request. Therefore, the value of U achieved by Algorithm 1 is lower than that achieved by Algorithm 2.

Figure 5.17: Impact of R on U

Because of the relationship between U and UU discussed earlier, only Algorithm 1 and NM display UU values lower than their respective U values. Algorithm 2 and NE have their UU values equal to their respective U (See Figure A-18 in Appendix A.6).

5.8 Impact of Error in User Estimation of Request Runtimes

5.8.1 Impact of Changing Parameters in Default Model of User Estimation Error

This section presents the impacts of different parameters of the default model of the error in user-estimation of the request runtimes used in the experiments. The description of the model is presented in Section 4.4.9.1.

5.8.1.1 Impact of the Percentage of Requests with Underestimated Runtimes caused by Users

In this set of experiments, the percentage of requests with underestimated runtimes in the workload (P_U) is varied from low to high in order to study its impact on the performance of the algorithms.

Figure 5.18 shows that changing values of P_U does not have any impact on the value of P_b achieved by NE because NE uses the actual runtimes of the requests.

Algorithm 1, Algorithm 2, and NM are not aware of the actual runtimes of the requests. As a result, a greater number of requests with underestimated runtimes in the workload allows the system using these algorithms to accept more requests. This explains why the algorithms have slightly lower P_b values when P_U increases.

Of the three algorithms, NM has the highest value of P_b as P_U increases. The reason is that the proportion of requests with underestimated runtimes is really small compared to the proportion of requests with overestimated runtimes in the workload. As NM does not make any adjustment to the estimated runtimes of the requests provided by

the users, a large proportion of requests with overestimated runtimes causes NM to reject many requests unnecessarily.

Figure 5.18: Impact of P_U on P_b

Algorithm 1 and Algorithm 2 compress the user-estimated runtimes of the requests and thus reduce the number of unnecessary rejections. Therefore, Algorithm 1 and Algorithm 2 have lower P_b values than NM.

In Figure 5.18, Algorithm 1 achieves a slightly lower P_b than Algorithm 2. As Algorithm 2 allows requests with underestimated runtimes caused by the users to run to completion, it increases the chances of other incoming requests being scheduled to start later. The overall effect is that the requests stay in the resource longer. This increases the chances of incoming requests overlapping with scheduled requests and being rejected by the system. In contrast, Algorithm 1 aborts requests with underestimated runtimes, helping to reduce the time these requests stay at the resources and increasing the chances of the resource schedule having more free slots for other incoming requests.

When P_U is low, the effect of not aborting requests with underestimated runtimes on the performance of Algorithm 1 versus Algorithm 2 is not significant. These algorithms have almost identical P_b values. However, the difference between the P_b values achieved by Algorithm 1 and Algorithm 2 increases when P_U is higher.

For reasons discussed earlier, A values also show a similar behavior as P_b when P_U increases (See Figure A-19 in Appendix A.7).

Figure 5.19 shows the effect of P_U on W_R for all algorithms. As NE does not have any runtime estimation error, W_R achieved by NE is not affected by changes in P_U . However, Algorithm 1, Algorithm 2, and NM accept more requests when there are more requests with underestimated runtimes in the workload. Thus, the amount of work that can be accepted on the system increases, and the W_R values of these algorithms decrease slightly with an increase in P_U .

As in the case of P_b , NM has the highest W_R while Algorithm 2 has a slightly higher W_R than Algorithm 1.

Figure 5.20 shows the effects of P_U on the values of U for all the algorithms. As NE does not have any changes in the values of W_R , its U values do not change either. Algorithm 1, Algorithm 2, and NM have reduced W_R when P_U increases. This allows these algorithms to have increased U values as more work is accepted into the system.

NM has the lowest U since it has the highest values of W_R . Algorithm 1 and Algorithm 2 have lower W_R , and thus higher U values, than NM. When P_U increases, Algorithm 1 displays a slightly lower U than Algorithm 2 even though it has a lower W_R than Algorithm 2. This is because Algorithm 1 aborts requests with underestimated

runtimes caused by users and thus prevents these requests from running up to their total runtimes while Algorithm 2 allows these requests to run to completion. This explains why Algorithm 2 has higher a U than Algorithm 1.

Figure 5.19: Impact of P_U on W_R

Figure 5.20: Impact of P_U on U

Figure 5.21 shows the effects of P_U on UU. Because NE and Algorithm 2 do not abort any requests, their UU for a given P_U is equal to their corresponding U values. Because Algorithm 1 and NM abort all requests with underestimated runtimes caused by users, their UU values decrease with an increase in P_U . With a greater number of requests with underestimated runtimes caused by the users in the workload (higher value of P_U), more requests with underestimated runtimes caused by users are aborted and thus reducing UU achieved by Algorithm 1 and NM.

Figure 5.21: Impact of P_U on UU

5.8.1.2 Impact of the Mean of Percentage of Error in Runtime Estimation for Abnormally Terminated Requests

This section studies the impact of changing the mean of the uniform distribution used for generating the percentage of runtime estimation error, denoted as η , of requests that are terminated abnormally. Since the runtimes used for the requests in the workload fit into only two runtime ranges defined in the error model (see Section 4.4.9.1), this section

changes the mean of uniform distribution used only in these two ranges. Table 7 lists the parameters of the uniform distributions used for generating the η for the requests whose runtimes fall into these two runtime ranges.

In this first set of experiments, the mean value of distribution U_1 (see Table 7), denoted as M_1 , is changed to determine if the performance of the algorithms is affected.

Table 7: Distribution details for abnormally terminated requests

Runtime range	Range of request runtimes (seconds)	Distribution for generating percentage error for these requests	Probability of requests being terminated abnormally in each runtime range
1	100 – 1,000	Uniform distribution U_1 (min= 10000%, max= 20000%)	0.09
2	1,000 – 10,000	Uniform distribution U_2 (min= 3000%, max= 10000%)	0.03

The experimental results show that changes to M_1 do not have any impact on the performance of the algorithms. This is because there are only a small number of requests that have their estimated runtimes provided by users change. As discussed in Section 4.4.8.2, the actual runtimes of the requests are generated using uniform distribution with a minimum value of 600 seconds and maximum value of 5,400 seconds. As a result, only requests having runtimes from 600 seconds to 1000 seconds fall into runtime range 1 in

Table 7. With request runtime uniformly distributed, the percentage of requests in the workload fall into range 1 is calculated to be $[(1000 - 600)/5400] * 100 = 8.3\%$. Among these requests, only 9% have η sampled from U_1 . Since the number of such requests is very small, this does not have any significant impact on the performance of the algorithms.

To save space, only graph of P_b is presented in Figure 5.22. Graphs of other performance metrics are shown in Figure A-20 to Figure A-23 in Appendix A.8.

Figure 5.22: Impact of M_1 on P_b

In the second set of experiments, the mean value for distribution U_2 (see Table 7), denoted as M_2 , is changed to determine if the performance of the algorithms is affected.

The results of these experiments are similar to those from the first set of experiments (see Figure 5.23). Once again, only the graph for P_b is shown here (see Figure 5.23). The graphs displaying results of other performance metrics are shown in Figure A-24 to Figure A-27 in Appendix A.8.

Figure 5.23: Impact of M_2 on P_b

5.8.1.3 Impact of the Mean of the Percentage Error in Runtime Estimation of Normally Terminated Requests

In this set of experiments, the mean of the hyper-exponential distribution used by the default user runtime estimation error model (described in Section 4.4.9.1) for generating η for normally terminated requests is varied. Since the runtimes used for requests in the workload fit into only two runtime ranges defined in the error model, the means of the hyper-exponential distribution used only in these two ranges are changed. The parameters of the hyper-exponential distributions are listed in Table 8.

In the first set of experiments, the mean value of distribution H_1 (see Table 8), denoted as HM_1 , is changed to determine if the performance of the algorithms is affected.

The experimental results show that changes to HM_1 do not have any significant impact on the performance of the algorithms (see Figure 5.24). The reason is the same as

provided in the context of the experiment involving a change M_1 discussed in Section 5.8.1.2.

Table 8: Distribution Details for Normally Terminated Requests

Runtime range	Range of request runtimes (seconds)	Distribution for generating percentage error for these requests	Probability of requests being terminated normally in each runtime range
1	100 – 1,000	Hyper-exponential (H_1) Mean = 1120%, CoV = 1.62	0.91
2	1,000 – 10,000	Hyper-exponential (H_2) Mean = 373%, CoV = 1.1	0.97

To save space, only graph of P_b is shown here in Figure 5.24. Graphs of other performance metrics are shown in Figure A-28 to Figure A-31 in Appendix A.9.

In the second set of experiments, the mean value of distribution H_2 (see Table 8), denoted as HM_2 , is changed to determine if the performance of the algorithms is affected.

Figure 5.25 shows the effect of HM_2 on the values of P_b for all the algorithms. P_b values appear to be fairly insensitive to changes in HM_2 .

A is based on P_b and the graph for A shows a similar trend as P_b . In addition, W_R shows similar behavior as P_b when HM_2 is varied. These metrics are presented in Figure A-32 and Figure A-33 in Appendix A.9.

Figure 5.24: Impact of HM_1 on P_b

Figure 5.26 shows that U values of NM decrease as HM_2 increases since P_b achieved with NM increases slightly as HM_2 increases. This is because NM uses the estimated runtimes of the requests provided by users for matchmaking. Increasing HM_2 means that request runtimes are overestimated by a higher amount. This causes NM to reject more requests. As the number of requests accepted to the system reduces, U achieved with NM reduces.

As HM_2 is increased, the difference between a request's deadline and its actual runtime tends to increase. As a result, NE that uses the actual request runtime can accept a higher number of requests and achieves a higher U .

The U values achieved with Algorithm 1 and Algorithm 2 seems to be relatively insensitive to a change in HM_2 .

Figure 5.25: Impact of HM_2 on P_b

Figure 5.26: Impact of HM_2 on U

As UU values for the algorithms are related to their corresponding U values, the UU values are included in Figure A-34 in Appendix A.9.

5.8.1.4 Impact of the Coefficient of Variation for Generating Percentage Error in Runtime Estimation of Normally Terminated Requests

This section studies the impact of the CoV of the hyper-exponential distribution used in the default model for error in user estimation of request runtimes (described in Section 4.4.9.1) on the performance of the algorithms. The hyper-exponential distribution is used to generate η (see Section 4.4.9.1) of all the requests that terminate normally. Based on the range of the request runtimes in the workload, there are only two distributions that need to have their CoV values changed. They are listed in Table 8 in Section 5.8.1.3.

In this first set of experiments, the CoV value of the distribution H_1 , denoted as CoV_1 , is changed to determine if the performance of the algorithms is affected.

The experimental results show that changes to CoV_1 do not have any significant impact on the performance of the algorithms. Once again, this is because there are only a small number of requests affected by changes to CoV_1 .

To save space, only graph for P_b is shown in Figure 5.27. Graphs for other performance metrics are shown in Appendix A.10, from Figure A-35 to Figure A-38.

In the second set of experiments, the CoV value of the distribution H_2 (CoV_2) is changed to determine if the performance of the algorithms is affected.

The experimental results show that changes to CoV_2 do not have any significant impact on the performance of the algorithms. Once again, this is because there are only a small number of requests affected by changes to CoV_2 .

Figure 5.27: Impact of CoV₁ on P_b

To save space, only the graph for P_b is shown in Figure 5.28. Graphs for other performance metrics are shown in Appendix A.10, from Figure A-39 to Figure A-42.

Figure 5.28: Impact of CoV₂ on P_b

5.8.2 Impact of Different Models of Error in User Estimation of Request Runtimes

This section presents the performance of the algorithms when different models of error in user estimation of request runtimes are used. The models for error in user runtime

estimation used in this set of experiments comprise: the constant model described in Section 4.4.9.2 and Feitelson's model described in Section 4.4.9.3. In this set of experiments, the constant error model is divided into two cases: one where runtimes of all requests are overestimated by 50% and another where the runtimes of all requests are underestimated by 50%. To determine the performance of the algorithms under different loads, the arrival rate of the workload is varied from low to high.

Figure 5.29 to Figure 5.31 show the P_b values for all algorithms when different models of error in user estimation of request runtimes are used. All these algorithms achieve comparable value of P_b at high λ when request runtimes are overestimated by a constant factor or when Feitelson's model is used (see Figure 5.29 and Figure 5.30). When Feitelson's model is used, as in Figure 5.29, NE has a lower P_b than Algorithm 1 and Algorithm 2. However, the difference in P_b between NE and Algorithm 1 and Algorithm 2 is still small compared to the difference in P_b between NM and NE.

For low and medium λ , Algorithm 1 and Algorithm 2 achieve comparable or lower P_b than NE when users either overestimate or underestimate all request runtimes by a constant factor, as shown in Figure 5.30 and Figure 5.31. When the request runtimes are underestimated by a constant factor, NE achieves highest P_b values while the other algorithms achieve statistically equivalent P_b values that are lower than that of NE at higher arrival rates (see Figure 5.31). Because all request runtimes are underestimated, NE that uses the actual request runtimes rejects a higher proportion of requests in comparison to the other algorithms.

Figure 5.29: P_b of all the algorithms when Feitelson's model is used

Figure 5.30: P_b of all the algorithms when all requests have overestimated runtimes

When runtimes of all requests are underestimated by a constant factor, NM has similar P_b values to Algorithm 1 and Algorithm 2 (see Figure 5.31). However, NM performs worse when there are requests with overestimated runtimes in the workload. Indeed, when the proportion of requests with overestimated runtimes in the workload is

high, as shown in Figure 5.29 and Figure 5.30, NM has the worst performance and obtains the highest P_b values among all the algorithms at low and medium λ .

Figure 5.31: P_b of all the algorithms when all requests have underestimated runtimes

As A and W_R values also display a similar behavior as P_b , the graphs of A and W_R are presented in Figure A-43 to Figure A-48 in Appendix A.11.

Figure 5.32 to Figure 5.34 show the U values of all the algorithms when different models of error in user estimation of request runtimes are used. When Feitelson's model is used, NE has lower P_b values than Algorithm 1 and Algorithm 2 (see Figure 5.29). This leads to NE having higher U values than Algorithm 1 and Algorithm 2, as indicated in Figure 5.32. When users always overestimate the request runtimes by a constant factor, Algorithm 1 and Algorithm 2 achieve statistically similar U values as NE (see Figure 5.33).

Figure 5.32: U of all the algorithms when Feitelson's model is used

Figure 5.33: U of all the algorithms when all requests have overestimated runtimes

Algorithm 1 aborts requests with underestimated runtimes whereas Algorithm 2 does not. Since the time spent serving these requests is less than the actual time required, the system using Algorithm 1 cannot achieve the same U value as the system using Algorithm 2. As a result, Algorithm 2 demonstrates higher values of U in comparison to

Algorithm 1 (see Figure 5.34). When the runtimes of all requests are underestimated by a constant factor, it cannot achieve the same U values as Algorithm 2 and NE (see Figure 5.34).

It is interesting to note that Algorithm 2 achieves a higher U in comparison to NE for all given value of λ . This is because all the request runtimes are underestimated and thus Algorithm 2 accepts a higher number of requests in comparison to NE that is based on the actual request runtimes. Since Algorithm 2 allows all requests to run to completion, it also demonstrates the best performances in terms of UU .

Figure 5.34: U of all the algorithms when all requests have underestimated runtimes

When all request runtimes are underestimated by a constant factor, NM achieves statistically similar P_b values as Algorithm 1 and Algorithm 2 (see Figure 5.31). Since NM aborts all requests, whose runtimes are underestimated by users, similar to Algorithm 1, its U values are similar to those of Algorithm 1 (see Figure 5.34).

NM performs worse than Algorithm 1 and Algorithm 2 when there are requests with overestimated runtimes in the workload. When the proportion of these requests in the workload is high, as in Figure 5.32 and Figure 5.33, NM has the lowest U values of all the other algorithms.

Figure 5.35 to Figure 5.37 show the UU values of all algorithms when different models of errors in user estimation runtimes are applied to the workload. Because Algorithm 2 and NE do not abort any requests, their UU and U values are similar.

Figure 5.35: UU of all the algorithms when Feitelson's model is used

Because Algorithm 1 and NM abort requests with runtimes that are underestimated by users, workloads containing these requests negatively affect their UU values. Figure 5.35 and Figure 5.37 confirm this observation. Figure 5.35 corresponds to a workload that has 10% of these requests. As a result, the UU values of Algorithm 1 and NM are lower than their U values for any given λ . The negative effect of requests with underestimated runtimes on the UU values of Algorithm 1 and NM becomes more

pronounced when the proportion of these requests in the workload is greater. The workload in Figure 5.37 contains requests with underestimated runtimes only; as a result, Algorithm 1 and NM abort all of the accepted requests, and thus, they both have a UU value of zero.

Figure 5.36: UU of all the algorithms when all requests have overestimated runtimes

Figure 5.37: UU of all the algorithms when all requests have underestimated runtimes

When the workload does not contain any requests with underestimated runtimes, as in Figure 5.36, both Algorithm 1 and NM have their UU values equal to their U values.

5.9 Impact of Request Runtimes

5.9.1 Impact of Mean Request Runtimes

This section presents the impact of mean request runtime (S) on the performance of the algorithms.

Figure 5.38 shows the impact of S on P_b . As S increases, the P_b values of all four algorithms increase. With higher S values, the requests are longer and thus stay at the resources for longer periods of time. This increases the chance of these requests overlapping with incoming requests and causing the incoming requests to be rejected.

Figure 5.38: Impact of S on P_b

NE achieves the lowest P_b with varying values of S due to no error in estimation of request runtimes. NM achieves the highest value of P_b because it relies on user

estimation of request runtimes for matchmaking. Because Algorithm 1 and Algorithm 2 compress the estimated runtime of requests, they avoid many unnecessary request rejections. As a result, Algorithm 1 and Algorithm 2 achieve lower values of P_b than NM.

As A is based on P_b , the graph for A values shows similar trend as P_b when S increases. W_R also shows a similar behavior as P_b . The graphs for these performance metrics are presented in Figure A-49 and Figure A-50 in Appendix A.12.

Figure 5.39 shows the values of U at different values of S . All algorithms show an increase in U as S increases.

Figure 5.39: Impact of S on U

As expected, NE achieves the highest U values and NM has the lowest U for any given value of S . Algorithm 1 and Algorithm 2 accept more requests than NM and therefore have higher U values than NM. However, Algorithm 1 has lower U values than Algorithm 2. This is because Algorithm 1 aborts requests with underestimated runtimes caused by users whereas Algorithm 2 does not.

As in the case of earlier experiments, UU for the algorithms is presented in Appendix A.12 (see Figure A-51).

5.9.2 Impact of Having a Big Range for Request Runtimes

This section presents the effect of using request runtimes that fit entirely into the runtime range of the model of error in user estimation of request runtimes described in Section 4.4.9.1. In other words, the request runtimes are sampled from a uniform distribution that spans *from 1 second to 100,000 seconds*. The arrival rate is varied in order to subject the system to different loads.

Figure 5.40 shows the value of P_b for different values of λ . Similar to the experiment presented in Section 5.1, all algorithms have increased P_b when λ is increased. NE has the lowest P_b and NM has the highest P_b values as expected. Algorithm 1 and Algorithm 2 still produce lower P_b than NM but the difference between the P_b values is not significant.

As A is based on P_b , the graph for A values shows a similar trend as P_b when S increases. W_R also shows a similar behavior as P_b . The graphs of these performance metrics are presented in Figure A-52 and Figure A-53 in Appendix A.12.

Figure 5.41 shows the values of U achieved with all algorithms when λ varies. For the same reasons presented in the experiment discussed in Section 5.1, all algorithms produce increased U when λ increases. NE, as expected, produces the highest U values and NM produces the lowest U values. Algorithm 1 and Algorithm 2 produce U values that are in between the values of U achieved with NE and NM.

Figure 5.40: Impact of having a big runtime range on P_b

Figure 5.41: Impact of having a big runtime range on U

As in the case of earlier experiments, UU for the algorithms is presented in Appendix A.13 (see Figure A-54).

5.9.3 Impact of Variability in Request Runtimes

This section studies the impact of the variability factor, denoted by V_f , for request runtime on the performance of the algorithms. V_f is used to change the maximum and minimum values of the uniform distribution used for generating request runtimes while keeping the mean of the uniform distribution for request runtime the same. In this set of experiments, the mean request runtimes is kept at the default value of 50 minutes. The upper and lower bounds of the uniform distribution for request runtimes are defined below.

$$\text{Minimum value} = \text{Mean request runtime} * (1 - V_f)$$

$$\text{Maximum value} = \text{Mean request runtime} * (1 + V_f)$$

Using a V_f of 0.1, 0.5, and 0.8 produces these uniform distributions: (45 minutes to 55 minutes), (25 minutes to 75 minutes), and (10 minutes to 90 minutes). These three ranges represent low variability, medium variability, and high variability in the request runtimes.

The experimental results show that the performance of the algorithms is relatively insensitive to the changes in values of V_f . To save space, only graph for P_b is presented in Figure 5.42. Graphs for other performance metrics are presented in Appendix A.14 (see Figure A-55 to Figure A-58).

5.9.4 Impact of High Variability in Request Runtimes

This section studies the impact of high variability in request runtimes on the performance of the algorithms. The request runtimes in this set of experiments are generated using

hyper-exponential distribution. The mean request runtime is kept at the default value of 50 minutes, as described in Section 4.4.8.2. The effect of the CoV of the hyper-exponential distribution used for generating request runtimes is of particular interest in these experiments.

Figure 5.42: Impact of V_f on P_b

Figure 5.43 shows the impact of CoV on the P_b values for all the algorithms. The P_b of NE decreases as CoV increases. As CoV increases, more requests with small runtimes are generated. Since the arrival rate does not change, a decrease in the length of requests allows the system to accept more requests. This explains why Algorithm 1, Algorithm 2, and NE have lower P_b values when CoV increases. As NM rejects a lot of requests, it is not sensitive to a change in CoV.

As A is derived from P_b , the graph for A values also shows a similar trend as P_b when CoV is increased (see Figure A-59 in Appendix A.15).

Figure 5.43: Impact of CoV on P_b

W_R for NM, Algorithm 1, and Algorithm 2 follow the same pattern as P_b presented in Figure 5.44. W_R for NE increases slightly with CoV. This is because more requests with small runtimes are generated with an increase in CoV, increasing the probability of NE accepting a greater number of small requests. Thus, NE tends to reject large requests as a result of which W_R increases.

U achieved with NM, Algorithm 1, and Algorithm 2 is insensitive to an increase in CoV (see Figure 5.45). The U value of NE decreases because W_R achieved with NE decreases as CoV increases.

As discussed in the previous cases, UU is less than U if the system aborts requests, and only Algorithm 1 and NM have UU values lower than their U values. Algorithm 2 and NE have their UU values equal to their respective U (See Figure A-60 in Appendix A.15).

Figure 5.44: Impact of CoV on W_R

Figure 5.45: Impact of CoV on U

5.10 Impact of the Resource Scheduling Policy

This set of experiments studies the impact of the scheduling policy at the resource on the performance of the algorithms. The scheduling policies used in the experiments are EDF,

ESF, and MLF. The descriptions of these scheduling algorithms are provided in Section 4.3.6.

The simulation results show that the performance of the algorithms is not affected by the local scheduling policy at the resource (see Figure 5.46 and Figure 5.47). To save space, this section shows only the graphs for P_b values for Algorithm 1 and Algorithm 2. The results for other performance metrics for all algorithms are shown in Appendix A.16 (see Figure A-61 to Figure A-78).

Figure 5.46: Impact of scheduling policy on P_b for Algorithm 1

5.11 Impact of Allocation Policy

This set of experiments studies the impact of allocation policy used by the broker on the performance of the algorithms. The allocation policies used in the experiments are FF and NF. The descriptions of these allocation algorithms are provided in Section 3.2.1.3.

Figure 5.47: Impact of scheduling policy on P_b for Algorithm 2

The simulation results show that the performance of the algorithms is not affected by the allocation policies used by the broker (see Figure 5.48 and Figure 5.49). To save space, this section shows only the graphs for P_b values for the Algorithm 1 and Algorithm 2 using different allocation policies. The results for other performance metrics for all algorithms are shown in Appendix A.17 (see Figure A-79 to Figure A-96).

5.12 Discussion

All the experimental results demonstrate that by allowing a small proportion of requests to miss their deadlines can significantly improve the performance of the system. In fact, UU achieved with Algorithm 1 is observed to increase by more than 65% as compared to UU achieved with NM at high λ (See Section 5.1). In addition, if requests whose runtimes are underestimated by users are allowed to run to completion, the performance gain is even more pronounced. Section 5.1 shows that at high λ , UU achieved with Algorithm 2 is increased by almost 100% as compared to UU achieved with NM that

aborts all these requests, at high λ . In addition, Algorithm 1 and Algorithm 2 always outperform NM regardless of which model for error in user estimation of request runtimes is used.

Figure 5.48: Impact of allocation policy on P_b for Algorithm 1

Figure 5.49: Impact of allocation policy on P_b for Algorithm 2

The experimental results provide guidance for choosing appropriate parameters for the SAR based matchmaking algorithms that allow the algorithms to perform effectively. The parameters are: p_{miss} , Δ , and W .

For the system and workload parameters experimented with, Section 5.6 shows that encouraging the users to choose p_{miss} to be 10% gives a good tradeoff between the performance of the algorithms and the proportion of requests missing their deadlines. p_{miss} less than 10% reduces the effectiveness of the algorithms. p_{miss} greater than 10% does not give any further performance improvement but reduces the incentives for the users to choose the cloud system using the SAR based matchmaking algorithms.

For the system and workload parameters experimented with, Δ equal to half of the value of p_{miss} allows the SAR based matchmaking algorithms to perform effectively (see Section 5.5). Δ greater than half of the value of p_{miss} reduces the performance of the algorithms. Choosing Δ to be less than half of the value of p_{miss} does not yield any more performance improvement for the algorithms. Investigating whether or not the value of p_{miss} that yields the tradeoff discussed in the previous paragraph and the corresponding value of Δ are sensitive to λ forms an interesting direction for future research.

For the system and workload parameters experimented with, 10 seems to be a reasonable value of W . W less than 10 reduces the performance that the algorithms can achieve. W greater than 10 does not yield any further performance improvement for the algorithms (see Section 5.4).

This thesis focuses on systems that are subjected to AR requests only. On systems that also accept OD requests that do not have deadlines, one can consider using the OD

requests for filling the gap in the resource schedule created when requests with overestimated runtimes complete before their expected completion times. This can potentially improve system utilization for NM, but P_b is likely to be reduced further because of accepting OD requests. This can lead to dissatisfaction of users submitting AR requests and can potentially reduce the revenue for the service provider because an AR request is expected to bring in more revenue in comparison to an OD request with the same execution time. This shows the effectiveness of the SAR based approach investigated in this thesis.

6 Chapter: Conclusions

This chapter provides a summary of the research performed in the thesis. Section 6.1 summarizes the algorithm for handling user-estimation error as well as the experimental set up for the simulation carried out in this thesis. Section 6.2 details insights into the system behavior and performance under different workloads and system parameters. Finally, a discussion of directions for future research is provided in Section 6.3.

6.1 Summary

In grids and clouds, two types of requests arrive on the systems: On-demand (OD) requests and Advance Reservation (AR) requests. On-demand requests are executed on a best-effort basis. Advance Reservation requests, in contrast, have deadlines, which must be met. Advance Reservation requests are used to provide a Quality of Service (QoS) guarantee to users.

Any-Schedulability (AS) criterion, described in Section 2.1, allows a broker to perform matchmaking for the AR requests without knowing the exact local scheduling policy at the resources. The only requirement for the criterion to be applicable is that the local scheduling policy at the resource must be work-conserving. Incorporating the AS criterion into the broker makes it possible to use resources without having a detailed *a priori* knowledge about the scheduling policies used at the resources.

This thesis investigates techniques to be deployed at an AS-based matchmaker for dealing with cases when there is error in user-estimated request runtimes. In fact, many studies report that the runtime supplied to the matchmaker by users is typically erroneous

[1], [13], [14]. In such cases, request runtime is either overestimated or underestimated; and in both cases, the performance of the broker is negatively affected.

To tackle the effect of user-estimation error in matchmaking, this thesis introduces the concept of Soft Advance Reservation (SAR), which can be used with the AS criterion. With SAR, the system allows a proportion of requests that arrive on the system to miss their deadlines; this proportion is called p_{miss} . The system tries to compress the estimated runtime of the request if the ratio of missed deadline requests up to that point is still less than p_{miss} . If the ratio of missed deadline requests is near or greater than p_{miss} , the user-estimated runtime is used instead. If the request is underestimated by the matchmaker, it is pre-empted and allowed to run to completion later, regardless of whether or not it misses its deadline. Based on the SAR concept, the thesis introduces two different algorithms: Algorithm 1 and Algorithm 2.

Algorithm 1 and Algorithm 2 compress the estimated runtime of the requests before applying the AS criterion for matchmaking. If the request's runtime is underestimated by the matchmaker, the algorithm allows it to run to completion. However, if the runtime is underestimated by the user, it is aborted. In contrast, Algorithm 2 allows all requests with underestimated runtimes to be pre-empted and run to completion at a later point in time, when there are free time slots available in the schedule.

To analyze the performance of the algorithms, extensive simulations were conducted. Various performance metrics were collected to determine the performance of the SAR based matchmaking algorithms. The main performance metrics were the

probability of blocking (P_b), acceptance ratio (A), percentage of work rejected (W_R), resource utilization (U), and resource useful utilization (UU) (see Section 4.1).

The performance of Algorithm 1 and Algorithm 2 was compared with those No-Error Algorithm (NE) and No-Modification Algorithm (NM). NE corresponds to a pure AS-based matchmaking when there is no error associated with the runtimes estimated by the matchmaker. NM also corresponds to a pure AS-based matchmaking but the estimated runtimes provided by the users are used without any modification.

6.2 Insights into System Behaviour and Performance

This section presents a number of important observations obtained from the simulation results.

6.2.1 Impact of Using Soft Advance Reservation

A set of key observations from the simulation results is summarized:

- The experimental results demonstrate the effectiveness of using Soft Advance Reservation to tackle errors in user estimation of request runtimes. Algorithm 1 and Algorithm 2 both outperform NM when the system is subjected to the same workload. Both of these algorithms lead to a higher useful utilization for resources that translates into higher revenues earned by the service providers and a lower P_b that is attractive to users because a higher proportion of the user requests get accepted.
- An efficient method using a linear compression scheme for modifying the user estimates of request runtimes (see Equation 3.4 in Section 3.2.1.2) was used. It

was observed to be effective in improving the system performance. Investigating other compression techniques in which the compression factor depends on the estimated runtimes for example can form an interesting direction for future research.

- Allowing requests with underestimated runtimes caused by users to run to completion is beneficial to the performance of the system. In fact, Algorithm 2 always achieves similar UU and U values for a given arrival rate whereas Algorithm 1 and NM have reduced UU values.
- Increasing p_{miss} allows Algorithm 1 and Algorithm 2 to have increased performance. As the system is allowed to have more missed deadline requests, it can compress the estimated runtime of more requests. More requests can be accepted as a result. For example, as shown in Figure 5.5, UU is observed to increase by more than 65% for Algorithm 1 and almost 100% for Algorithm 2 at high arrival rate (λ equal to 0.011 requests per second) when only 10% requests are allowed to miss deadlines. Similarly, Figure 5.1 shows that P_b decreases by more than 10% at low and medium arrival rate when p_{miss} is set at 10%. Increasing p_{miss} helps the performance only if it allows more requests to be compressed. After a certain value of p_{miss} is reached, P_b , A, W_R , U, and UU achieved with Algorithm 1 and Algorithm 2 seem to flatten off.

6.2.2 Comparison of Algorithm 1 and Algorithm 2

The SAR based approach that allows a small proportion of requests to miss their deadlines was discussed in this thesis. Two algorithms, Algorithm 1 and Algorithm 2 that

can be used in conjunction with SAR, were investigated. A simulation based analysis of the two algorithms was carried out for a number of combinations of system and workload parameters.

Algorithm 1 and Algorithm 2 achieve statistically comparable P_b values. However, Algorithm 2 outperforms Algorithm 1 in terms of UU. This indicates resources in systems using Algorithm 2 are utilized more than resources in systems using Algorithm 1. As users are charged based on how long their requests use the resource, Algorithm 2 is expected to bring more revenue in for the service providers as compared to Algorithm 1.

6.2.3 Impact of System and Workload Parameters on Performance

A number of workload and system parameters were experimented with. Insights gained into system behavior and performance from the results of these experiments is presented.

- An increased number of resources available to the system leads to an increase in system performance for NE and NM. With more resources available, the system can accept more requests. Algorithm 1 and Algorithm 2 have increased A, U, and UU when the number of resources increases up to a certain value. When the number of resources increases over that value, the A, U, and UU achieved with both Algorithm 1 and Algorithm 2 flatten off. The relative performances of the algorithms are expected to remain the same for other values of R.
- The local scheduling policies used at the resources do not have a statistically significant impact on the performance of the algorithms.

- Similarly, the allocation policy employed by the matchmaker does not seem to have a significant impact on the performance of the algorithms.
- Increasing the mean runtime of requests causes all algorithms to have reduced A values and increased P_b , W_R , U , and UU values. This is because higher mean request runtimes increase the amount of work subjected to the system. As the system has a limited capacity, it tends to reject more requests as the mean runtime of requests increases. However, the resource utilization of the system increases since requests may be fitted better into the resource schedule.
- Increasing laxities of requests leads to an increased performance of the algorithms. As requests have more laxity, the system has more flexibility in scheduling requests. This leads to an increase in the proportion of requests accepted into the system.
- Increases in T_s lead to requests staying longer at the resources. As a result, with NE and NM, more requests are rejected and resource utilization decreases. However, for Algorithm 1 and Algorithm 2, there are two observations. Low T_s (lower than 12 hours) causes requests to start sooner and thus increases contention at the resources. An increase in T_s (up to 12 hours) spreads out the start time of requests and allows more flexibility for the scheduler to order the requests. Thus, P_b and W_R achieved with Algorithm 1 and Algorithm 2 decrease while A , U , and UU increase. Increases in T_s beyond 12 hours cause requests to stay at the resource longer, and tend to increase the chance of overlap with incoming

requests. This causes the P_b and W_R of these algorithms to increase and A , U , and UU decrease accordingly.

- Irrespective of which model for error in user estimation of request runtimes is used, Algorithm 1 and Algorithm 2 always outperform NM. When all requests have runtimes overestimated by a constant percentage by the users, Algorithm 1 and Algorithm 2 achieve a similar performance as compared to NE.

6.3 Future Research Directions

A number of issues that are worthy of investigation in the future are presented in this section.

- The system investigated in this thesis is characterized by a single p_{miss} . The value of p_{miss} can be negotiated as an SLA parameter with a specific user. Investigating systems that allow multiple p_{miss} values, one for a set of requests submitted by a specific user forms an interesting direction for future research.
- In the solution investigated in this thesis, requests are treated equally without any regard to their priority. On certain systems, requests may have different priorities and higher priority requests are expected to experience a lower blocking in comparison to lower priority requests. In addition, these requests are expected to experience a lower probability of being aborted if their runtimes are underestimated. A modified algorithm that allows higher priority requests to be accepted at the cost of causing a small proportion of lower priority requests to miss their deadlines while ensuring that the proportion of all requests missing

their deadlines remains lower than the prescribed p_{miss} value is worthy of future research.

- Workflow is an important type of workload in grids and clouds where all the tasks belong to a workflow exhibit dependency among one another. Investigating the performance of SAR for workflow tasks in grids and clouds is worth considering.
- The resources in the system are considered to be static and resources do not leave the system at any point in time. As resources on certain systems may leave the grid/cloud, requests running on these resources may have to be restarted somewhere else. This represents a significant delay which increases the chance for these requests to miss their deadlines. Investigating the performance of the SAR based matchmaking solution in this scenario is worthy of future research.
- The experimental results in this research are obtained using a simulation environment where the network delay is assumed to be negligible. An investigation into the performance of algorithms using real workloads and real network delays is a worthy direction for future research.

References

- [1] C. Bailey Lee, Y. Schwartzman, J. Hardy, and A. Snaveley, “Are User Runtime Estimates Inherently Inaccurate?,” in *Job Scheduling Strategies for Parallel Processing*, vol. 3277, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 253–263.
- [2] G. Birkenheuer, A. Brinkmann, and H. Karl, “Risk aware overbooking for commercial grids,” in *Proceedings of the 15th international conference on Job scheduling strategies for parallel processing*, Berlin, Heidelberg, 2010, pp. 51–76.
- [3] R. Buyya and M. Murshed, “GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing,” *Concurrency and Computation: Practice and Experience (CCPE)*, vol. 14, no. 13, pp. 1175–1220, 2002.
- [4] H. Casanova, F. Berman, G. Obertelli, and R. Wolski, “The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid,” in *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, 2000, pp. 60– 79.
- [5] X. Che, L. Hu, D. Guo, K. Tang, and D. Hu, “Information Service Prototype System for Run-time Prediction of Grid Applications,” in *Proceedings of the 2nd International Conference on Pervasive Computing and Applications (ICPCA 2007)*, 2007, pp. 530 –535.
- [6] S. Chiang, A. Arpaci-Dusseau, and M. K. Vernon, “The Impact of More Accurate Requested Runtimes on Production Job Scheduling Performance,” in *Job Scheduling Strategies for Parallel Processing*, vol. 2537, D. G. Feitelson, L.

- Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 103–127.
- [7] L. Chunlin and L. Layuan, “Optimization decomposition approach for layered QoS scheduling in grid computing,” *Journal of Systems Architecture*, vol. 53, no. 11, pp. 816–832, Nov. 2007.
 - [8] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.
 - [9] P. A. Dinda, “The statistical properties of host load,” *Scientific Programming*, vol. 7, no. 3–4, pp. 211–229, Aug. 1999.
 - [10] M. Dorigo, M. Birattari, and T. Stutzle, “Ant colony optimization,” *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28–39, Nov. 2006.
 - [11] E. Elghoneimy, O. Bouhali, and H. Alnuweiri, “Resource allocation and scheduling in cloud computing,” in *Proceedings of the 2012 International Conference on Computing, Networking and Communications (ICNC)*, 2012, pp. 309–314.
 - [12] D. England, J. Weissman, and J. Sadagopan, “A new metric for robustness with application to job scheduling,” in *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*, 2005, pp. 135–143.
 - [13] U. Farooq, “A Framework for Quality of Service Aware Resource Management in Multi-Institutional Grids,” PhD. Thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada, 2007.

- [14] U. Farooq, S. Majumdar, and E. W. Parsons, "Achieving efficiency, quality of service and robustness in multi-organizational Grids," *Journal of Systems and Software*, vol. 82, pp. 23–38, Jan. 2009.
- [15] D. G. Feitelson, "Lublin/Feitelson Workload Model, C Code." [Online]. Available: <http://www.cs.huji.ac.il/labs/parallel/workload/models.html#lublin99>. [Accessed: 18-Apr-2011].
- [16] I. Foster, "What is the Grid? A Three Point Checklist," *Grid Today*, vol. 1, no. 6, pp. 22–25, 2002.
- [17] I. Foster, "The anatomy of the grid: enabling scalable virtual organizations," in *Proceedings of the 1st IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2001, pp. 6–7.
- [18] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava, "Building a high-level dataflow system on top of Map-Reduce: the Pig experience," *Proceedings of the Very Large Database Endowment (VLDB)*, vol. 2, no. 2, pp. 1414–1425, Aug. 2009.
- [19] R. Gibbons, "A Historical Application Profiler for Use by Parallel Schedulers," in *Proceedings of the Job Scheduling Strategies for Parallel Processing*, London, UK, 1997, pp. 58–77.
- [20] GridSim, "The CLOUDS Lab: Flagship Projects - Gridbus and Cloudbus." [Online]. Available: <http://www.cloudbus.org/gridsim/>. [Accessed: 31-Aug-2011].

- [21] T. J. Hacker and B. D. Athey, "A Methodology for Account Management in Grid Computing Environments," in *Proceedings of the 2nd International Workshop on Grid Computing*, London, UK, 2001, pp. 133–144.
- [22] B. Jacob, "Grid computing: What are the key components? Taking advantage of grid computing for application enablement." IBM's Developer Networks Grid Library, IBM Corporation, Jun-2003.
- [23] B. Jacob and V. Berstis, "IBM Redbooks | Fundamentals of Grid Computing." [Online]. Available: <http://www.redbooks.ibm.com/abstracts/redp3613.html>. [Accessed: 24-Feb-2012].
- [24] M. Kiran, A. H. Abdalla, Y. Y. Jiun, and L. M. Kuan, "A prediction module to optimize scheduling in a grid computing environment," in *Proceedings of the International Conference on Computer and Communication Engineering (ICCCCE 2008)*, 2008, pp. 888 –893.
- [25] M. Kretzschmar and S. Hanigk, "Security management interoperability challenges for Collaborative Clouds," in *Proceedings of the 4th International DMTF Academic Alliance Workshop on Systems and Virtualization Management (SVM)*, 2010, pp. 43 –49.
- [26] G. Lee, N. Tolia, P. Ranganathan, and R. H. Katz, "Topology-aware resource allocation for data-intensive workloads," *SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 120–124, Jan. 2011.
- [27] B. Li, J. Chen, M. Yang, and E. Wang, "Impact of Extending the Runtime of Underestimated Jobs in Backfilling Schedulers," in *Proceedings of the 2008*

- International Conference on Computer Science and Software Engineering*, 2008, vol. 3, pp. 328–331.
- [28] K. Li, G. Xu, G. Zhao, Y. Dong, and D. Wang, “Cloud Task Scheduling Based on Load Balancing Ant Colony Optimization,” in *Proceedings of the 6th Annual Chinagrid Conference (ChinaGrid)*, 2011, pp. 3 –9.
- [29] S. Majumdar, “The ‘Any-Schedulability’ Criterion for Providing QoS Guarantees through Advance Reservation Requests,” in *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid '09)*, 2009, pp. 490 –495.
- [30] A. Matsunaga and J. Fortes, “On the Use of Machine Learning to Predict the Time and Resources Consumed by Applications,” in *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, 2010, pp. 495 –504.
- [31] J. O. Melendez, “Matchmaking on Distributed Systems with Limited Knowledge of Resources,” M.A.Sc. Thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada, 2010.
- [32] J. O. Melendez and S. Majumdar, “Matchmaking with limited knowledge of resources on clouds and grids,” in *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, 2010, pp. 102–110.
- [33] J. O. Melendez and S. Majumdar, “Utilizing ‘Opaque’ Resources for Revenue Enhancement on Clouds and Grids,” in *Proceedings of the 11th IEEE/ACM*

- International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2011, pp. 576–584.
- [34] I. Moschakis and H. Karatza, “Evaluation of gang scheduling performance and cost in a cloud computing system,” *The Journal of Supercomputing*, vol. 59, no. 2, pp. 975–992, 2012.
- [35] A. W. Mu’alem and D. G. Feitelson, “Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. 12, no. 6, pp. 529–543, Jun. 2001.
- [36] Y. Murata, R. Egawa, M. Higashida, and H. Kobayashi, “A History-Based Job Scheduling Mechanism for the Vector Computing Cloud,” in *Proceedings of the 10th IEEE/IPSJ International Symposium on Applications and the Internet (SAINT)*, 2010, pp. 125–128.
- [37] NIST, “A NIST definition of cloud computing.” [Online]. Available: csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf. [Accessed: 20-Jun-2012].
- [38] A. Oprescu and T. Kielmann, “Bag-of-Tasks Scheduling under Budget Constraints,” in *Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2010, pp. 351–359.
- [39] M. Paul and G. Sanyal, “Survey and analysis of optimal scheduling strategies in cloud environment,” in *Proceedings of the 2011 World Congress on Information and Communication Technologies (WICT)*, 2011, pp. 789–792.

- [40] W. Smith, I. Foster, and V. Taylor, "Scheduling with advanced reservations," in *Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS 2000)*, 2000, pp. 127–132.
- [41] S. Sotiriadis, N. Bessis, and N. Antonopoulos, "Towards Inter-cloud Schedulers: A Survey of Meta-scheduling Approaches," in *Proceedings of the 2011 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, 2011, pp. 59–66.
- [42] A. Sulistio and R. Buyya, "A Grid Simulation Infrastructure Supporting Advance Reservation," in *Proceedings of the 16th International Conference on Parallel and Distributed Computing and Systems (PDCS 2004)*, MIT Cambridge, Boston, USA, 2004.
- [43] W. Tang, N. Desai, D. Buettner, and Z. Lan, "Analyzing and adjusting user runtime estimates to improve job scheduling on the Blue Gene/P," in *Proceedings of the 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, 2010, pp. 1–11.
- [44] L. Tomás, A. C. Caminero, C. Carrión, and B. Caminero, "Network-aware meta-scheduling in advance with autonomous self-tuning system," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 486–497, May 2011.
- [45] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling Using System-Generated Predictions Rather than User Runtime Estimates," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 18, no. 6, pp. 789–803, 2007.

- [46] X. Wang, Y. Xue, L. Fan, R. Wang, and Z. Du, "Research on Adaptive QoS-Aware Resource Reservation Management in Cloud Service Environments," in *Proceedings of the 2011 IEEE Asia-Pacific on Services Computing Conference (APSCC)*, 2011, pp. 147–152.
- [47] J. Weinman, "The future of Cloud Computing," in *Proceedings of the 2011 IEEE Technology Time Machine Symposium on Technologies Beyond 2020 (TTM)*, 2011, pp. 1–2.
- [48] A. K. L. Wong and A. M. Goscinski, "The Impact of Under-Estimated Length of Jobs on EASY-Backfill Scheduling," in *Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)*, 2008, pp. 343–350.
- [49] P. Xiao and Z. Hu, "Relaxed resource advance reservation policy in grid computing," *The Journal of China Universities of Posts and Telecommunications*, vol. 16, no. 2, pp. 108–113, Apr. 2009.
- [50] Y. Yuan and W. Liu, "Efficient resource management for cloud computing," in *Proceedings of the 2011 International Conference on System Science, Engineering Design and Manufacturing Informatization (ICSEM)*, 2011, vol. 2, pp. 233–236.
- [51] S. Zhang, S. Zhang, X. Chen, and X. Huo, "Cloud Computing Research and Development Trend," in *Proceedings of the 2nd International Conference on Future Networks (ICFN'10)*, 2010, pp. 93–97.

- [52] Z. Zhang, L. Cherkasova, A. Verma, and B. T. Loo, "Meeting service level objectives of Pig programs," in *Proceedings of the 2nd International Workshop on Cloud Computing Platforms*, New York, NY, USA, 2012, pp. 8:1–8:6.
- [53] "Type of grids." [Online]. Available: <http://www.it-tude.com/types-of-grids.html>. [Accessed: 09-Feb-2012].
- [54] "Amazon Elastic Compute Cloud (Amazon EC2)." [Online]. Available: <http://aws.amazon.com/ec2/>. [Accessed: 25-Feb-2012].
- [55] "Amazon Simple Storage Service (Amazon S3)." [Online]. Available: <http://aws.amazon.com/s3/>. [Accessed: 09-Feb-2012].
- [56] "Haizea - An Open Source VM-based Lease Manager." [Online]. Available: <http://haizea.cs.uchicago.edu/>.
- [57] "Google App Engine." [Online]. Available: <http://code.google.com/appengine/>. [Accessed: 25-Feb-2012].

Appendix

Appendix A

This section presents the results that have not shown in Chapter 5.

A.1 Impact of T_s

Figure A-1: Impact of T_s on A

Figure A-2: Impact of T_s on W_R

Figure A-3: Impact of T_s on UU

A.2 Impact of Laxity (L)

Figure A-4: Impact of L on A

Figure A-5: Impact of L on W_R

Figure A-6: Impact of L on UU

A.3 Impact of Window Size (W)

Figure A-7: Impact of W on A

Figure A-8: Impact of W on W_R

Figure A-9: Impact of W on UU

A.4 Impact of Margin (Δ)

Figure A-10: Impact of Δ on A

Figure A-11: Impact of Δ on W_R

Figure A-12: Impact of Δ on UU

A.5 Impact of the Proportion of Allowable Missed Deadline (p_{miss})

Figure A-13: Impact of p_{miss} on A

Figure A-14: Impact of p_{miss} on W_R

Figure A-15: Impact of p_{miss} on UU

A.6 Impact of Number of Resources (R)

Figure A-16: Impact of R on A

Figure A-17: Impact of R on W_R

Figure A-18: Impact of R on UU

A.7 Impact of the Percentage of Requests with Runtimes Underestimated by Users

Figure A-19: Impact of P_U on A

A.8 Impact of the Mean of Percentage of Error in User Estimation of Request Runtimes for Abnormally Terminated Requests

Figure A-20: Impact of M_1 on A

Figure A-21: Impact of M_1 on W_R

Figure A-22: Impact of M_1 on U

Figure A-23: Impact of M_1 on UU

Figure A-24: Impact of M_2 on A

Figure A-25: Impact of M_2 on W_R

Figure A-26: Impact of M_2 on U

Figure A-27: Impact of M_2 on UU

A.9 Impact of the Mean of the Percentage of Error in User Estimation of Request

Runtimes for Normally Terminated Requests

Figure A-28: Impact of HM_1 on A

Figure A-29: Impact of HM₁ on W_R

Figure A-30: Impact of HM₁ on U

Figure A-31: Impact of HM₁ on UU

Figure A-32: Impact of HM₂ on A

Figure A-33: Impact of HM₂ on W_R

Figure A-34: Impact of HM₂ on UU

A.10 Impact of the Coefficient of Variation of the Hyper-exponential Distribution

Figure A-35: Impact of CoV_1 on A

Figure A-36: Impact of CoV_1 on W_R

Figure A-37: Impact of CoV₁ on U

Figure A-38: Impact of CoV₁ on UU

Figure A-39: Impact of CoV_2 on A

Figure A-40: Impact of CoV_2 on W_R

Figure A-41: Impact of CoV₂ on U

Figure A-42: Impact of CoV₂ on UU

A.11 Impact of Different Models of Error in User Estimation of Request Runtimes

Figure A-43: A values of all the algorithms when Feitelson's model is used

Figure A-44: A values of all the algorithms when all requests have overestimated runtimes

Figure A-45: A values of all the algorithms when all requests have underestimated runtimes

Figure A-46: W_R of all the algorithms when Feitelson's model is used

Figure A-47: W_R of all the algorithms when all requests have overestimated runtimes

Figure A-48: W_R of all the algorithms when all requests have underestimated runtimes

A.12 Impact of Mean Request Runtimes

Figure A-49: Impact of S on A

Figure A-50: Impact of S on W_R

Figure A-51: Impact of S on UU

A.13 Impact of Having a Big Runtime Range

Figure A-52: Impact of having a big runtime range on A

Figure A-53: Impact of having a big runtime range on W_R

Figure A-54: Impact of having a big runtime range on UU

A.14 Impact of Variability in Request Runtimes

Figure A-55: Impact of V_f on A

Figure A-56: Impact of V_f on W_R

Figure A-57: Impact of V_f on U

Figure A-58: Impact of V_f on UU

A.15 Impact of High Variability in Request Runtime

Figure A-59: Impact of CoV on A

Figure A-60: Impact of CoV on UU

A.16 Impact of Scheduling Policy

A.16.1 Scheduling Policy Impact for Algorithm 1

Figure A-61: Impact of scheduling policy on A for Algorithm 1

Figure A-62: Impact of scheduling policy on W_R for Algorithm 1

Figure A-63: Impact of scheduling policy on U for Algorithm 1

Figure A-64: Impact of scheduling policy on UU for Algorithm 1

A.16.2 Scheduling Policy Impact for Algorithm 2

Figure A-65: Impact of scheduling policy on A for Algorithm 2

Figure A-66: Impact of scheduling policy on W_R for Algorithm 2

Figure A-67: Impact of scheduling policy on U for Algorithm 2

Figure A-68: Impact of scheduling policy on UU for Algorithm 2

A.16.3 Scheduling Policy Impact for NE

Figure A-69: Impact of scheduling policy on P_b for NE

Figure A-70: Impact of scheduling policy on A for NE

Figure A-71: Impact of scheduling policy on W_R for NE

Figure A-72: Impact of scheduling policy on U for NE

Figure A-73: Impact of scheduling policy on UU for NE

A.16.4 Scheduling Policy Impact for NM

Figure A-74: Impact of scheduling policy on P_b for NM

Figure A-75: Impact of scheduling policy on A for NM

Figure A-76: Impact of scheduling policy on W_R for NM

Figure A-77: Impact of scheduling policy on U for NM

Figure A-78: Impact of scheduling policy on UU for NM

A.17 Impact of Allocation Policy

A.17.1 Allocation Policy Impact on Algorithm 1

Figure A-79: Impact of allocation policy on A for Algorithm 1

Figure A-80: Impact of allocation policy on W_R for Algorithm 1

Figure A-81: Impact of allocation policy on U for Algorithm 1

Figure A-82: Impact of allocation policy on UU for Algorithm 1

A.17.2 Allocation Policy Impact on Algorithm 2

Figure A-83: Impact of allocation policy on A for Algorithm 2

Figure A-84: Impact of allocation policy on W_R for Algorithm 2

Figure A-85: Impact of allocation policy on U for Algorithm 2

Figure A-86: Impact of allocation policy on UU for Algorithm 2

A.17.3 Allocation Policy Impact on NE

Figure A-87: Impact of allocation policy on P_b for NE

Figure A-88: Impact of allocation policy on A for NE

Figure A-89: Impact of allocation policy on W_R for NE

Figure A-90: Impact of allocation policy on U for NE

Figure A-91: Impact of allocation policy on UU for NE

A.17.4 Allocation Policy Impact on NM

Figure A-92: Impact of allocation policy on P_b for NM

Figure A-93: Impact of allocation policy on A for NM

Figure A-94: Impact of allocation policy on W_R for NM

Figure A-95: Impact of allocation policy on U for NM

Figure A-96: Impact of allocation policy on UU for NM