Techniques for Co-Allocation of Resources in Grids Processing Advance Reservation Requests

By

Kirushananth Thirukailayanathan, B.Eng. (SCE)

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirement for the degree of

Master of Applied Science in Electrical Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Faculty of Engineering

Carleton University

Ottawa, Ontario, Canada

May 2011

Your file Votre référence
ISBN: 978-0-494-83031-4
Our file Notre référence
ISBN: 978-0-494-83031-4

The undersigned hereby recommend to

The Faculty of Graduate Studies and Postdoctoral Affairs

the acceptance of the thesis

Techniques for Co-Allocation of Resources in Grids Processing Advance Reservation Requests

Submitted by

Kirushananth Thirukailayanathan, B.Eng.

in partial fulfillment of the requirements for the degree of

Master of Applied Science in

Electrical and Computer Engineering

Shikharesh Majumdar

Thesis Co-Supervisor, Department of Systems and Computer Engineering

---

Biswajit Nandy

Thesis Co-Supervisor, Department of Systems and Computer Engineering

---

Howard Schwartz

Chair, Department of Systems and Computer Engineering

Carleton University

May 2011

Abstract

In grids, resource co-allocation is required in situations where applications require simultaneous availability of multiple resources throughout their execution period. Simultaneous availability of the required resources within the time period of start time and end time of the application is essential for successful completion of the application. The selection of appropriate resources for co-allocation so that the desired resource utilization is achieved for the resource providers as well as the quality of service requirements of the resource consumers are satisfied, is performed by the co-allocation algorithm in a broker. This research presents co-allocation algorithms that take advantage of the different steps in the co-allocation process for ensuring effectiveness of the algorithms. Performance analysis of the co-allocation algorithm is performed using simulations. A thorough analysis of the simulation results and key observations in terms of system performance are presented in this thesis.

Acknowledgements

Firstly, I would like to express my sincere thanks to my supervisors Prof. Shikharesh Majumdar and Prof. Biswajit Nandy for their guidance, continuous support, encouragement and help throughout the thesis. I deeply appreciate your guidance and patience during each phase of the thesis.

I would like to thank my peers Navdeep and Orlando for their help on getting me started with Globus Toolkit.

I am thankful to my family and friends for their continuous support and encouragement throughout the thesis.

Table of Contents

List of Tables

List of Figures

List of Symbols and Acronyms

λ	Arrival Rate
2PC	Two-Phased Commit
AR	Advance Reservation
API	Application Provider Interface
B	Blocking Ratio
EFT	Earliest-Finish-Time
FCFS	First-Come-First-Served
GARA	Globus Architecture for Reservation and Allocation
GARS	Grid Advance Reservation-based Scheduling Framework
GIIS	Globus Index Information Service
GIS	Grid Information Service
GMA	Grid Monitoring Architecture
GridARS	Grid Advance Reservation-based System
GridSAM	Grid Job Submission and Monitoring Web Service
GRIS	Globus Resource Information Service
GSI	Grid Security Infrastructure
HARC	Highly-Available Robust Co-scheduler
QoS	Quality of Service
Q_R	Fairness
MDS	Globus Monitoring and Discovery System
RMS	Resource Management System

RSVP	Resource Reservation Protocol
SPI	Service provider Interfaces
WSRF	Web Services resource Framework
U	Utilization
W_R	Percentage of Work Rejected

Chapter 1
Introduction

1.1 Background

In recent years, demands for high performance computational capabilities, extensive communication and data usage, has been ever increasing. The size and complexity of applications increase as advances in technology add more computational capabilities to hardware. Applications become more sophisticated and their requirements often exceed the resource capabilities available within a single organization. Maintaining dedicated high resource capabilities within a single organization and to accommodate the increasing resource demands of the applications is very costly. To accommodate these changing needs and challenges, organizations have moved from a centralized computing architecture towards a distributed computing architecture. Distributed computing provides a more scalable solution for the problems that the resource intensive applications produce. However, the scalability in distributed computing is limited by the number of resources available at the organizations. The need for sharing of resources that are geographically distributed becomes prominent to overcome these scalability problems. Advances in Internet technologies and high-speed networking capabilities allow the geographically distributed resources to become accessible for distributed computing with minimal cost.

Grid computing provides an effective and scalable distributed solution for applications with high demands for computing, communication and storage. Instead of dedicating resources for applications within the administrative domain, grid computing enables the sharing, selection and aggregation of a pool of independently controlled and administrated resources available in

multiple organizations. A set of grid standards allows secured and coordinated access of heterogeneous resources that are geographically distributed and reside in different administrative domains [FOS-02]. The heterogeneous nature of the grid resources and the fact that they are independently controlled and administrated by different administrative domains provides many challenges for both grid resource consumers and providers to overcome. Resource providers need to ensure that their resources are shared by resource consumers so that an effective utilization of resource and revenue is achieved. Resource consumers need to ensure that required resources are available for their applications and that these resources meet their requirements. The common requirements of grid applications include, meeting quality of service requirements, required resource capabilities, and availability of multiple resources for parallel tasks.

This thesis explores the challenges in co-allocating resources for applications that consist of parallel tasks that need to be co-allocated on multiple resources so that they can run simultaneously. These applications require all resources and their communication links between one another to be available throughout the execution period of these applications. The diverse nature of these resources, network communication latencies and different scheduling policies from different administrative domains, makes the co-allocation of these resources a challenging task for resource brokers. Inefficient co-allocation of resources may result in poor utilization of resources. There have been several research studies done in the context for co-allocation of resources.

1.2 Motivation for the Thesis

Grid systems provide effective, scalable and flexible environments for grid applications, by enabling sharing, selection, and aggregation of a wide range of distributed resources that are independently controlled and administrated. Grid applications are generally resource intensive and have high demands for memory, storage, and processing power. Parallel computing has been one of the essential requirements for high performance grid applications. Some of the examples of such applications are: data intensive communication using multiple channels, resource intensive multimedia applications, scientific research computations and geo-processing. Available resources from a single administrative domain often fail to fulfill the increasing demands of application requirements, thus forcing the design and deployment of such applications to move towards grid computing [JOS-1].

Allocating resources for applications that require parallel computing, is a challenging task even in a single multiprocessor system. These applications require guaranteed simultaneous availability of all the required resources throughout the execution period of the applications. Unavailability of a single required resource during the execution of the application might compromise the execution of the entire application. The diverse nature of resources, network communication latencies and different scheduling policies from different administrative domains, makes the resource co-allocation process more challenging in grid environments. Advance reservation solves these problems by reserving the required resources in advance to guarantee their availability at the scheduled start time and during the execution period of the application. Advance reservations of resources also help to provide quality of service guarantees to the resource consumer.

Choosing the appropriate resources for co-allocation plays a key role in meeting the resource provider's goals of optimized resource utilization. Since co-allocation involves reservation of multiple resources to ensure the simultaneous availability, special care has to be given when selecting the resources. Reserving the required resources in advance without the considerations of dynamic characteristics of the resources and in an uncontrolled manner, may downgrade system performance [SID-06]. Appropriate control of advance reservation of resources is necessary for meeting the objectives of both resource providers and consumers. This thesis focuses on co-allocation strategies that can help the resource providers to achieve optimized system performance while providing QoS guarantees to resource consumers.

In recent years, cloud computing has emerged as a new and promising paradigm for providing computational resources based on Service Level Agreements (SLA) to computing service users (consumers) [BUY-09]. Cloud computing offers services making resources available on demand to consumers and enables users to increase or decrease the resource capacity they consume based on their requirements [ARM-09]. Resource management in cloud computing share common grounds with grid computing and existing techniques for resource management on grids can be extended to cloud computing. Co-allocation techniques presented in this research can also be applied in cloud computing environments to serve applications that demand simultaneous allocations of multiple resources.

1.3 Contributions of the Thesis

The main contributions of the thesis are listed.

- New co-allocation algorithms that explore various ways to provide enhanced system performance to resource providers while ensuring quality of service for resource consumers.
- The co-allocation algorithms introduced in the thesis were compared with an algorithm that evaluates each possible resource combination for allocation, In spite of having a lower complexity in comparison to this algorithm that performs an exhaustive search in the solution space, the co-allocation algorithms introduced in this thesis provided a comparable performance for a range of parameter values experimented with.
- An understanding of what information regarding system state leads to a high performing algorithm.
- Insights into effects of various workload and system parameters on the performance of co-allocation algorithms. These insights are based on the results of running a simulator that was devised for this research.
- Comparisons of the proposed algorithm for determining the best and worst strategies for selecting resources and their impacts on system performance.

1.4 Thesis Outline

The thesis comprises six chapters. Chapter 2 provides an overview of grid computing, grid resource access and management and related work on resource co-allocation. Chapter 3 gives a detailed view of resource co-allocation and proposes new algorithms to solve the co-allocation problems. The simulation model is explained in chapter 3, providing details about experimental setup, workload and system parameters and performance metrics. Chapter 4 discusses the

performance results of the proposed algorithms for various workloads and presents a comparison of the algorithms. Chapter 5 concludes the thesis and discusses directions for future research.

Chapter 2
Background

2.1. Overview of Grid Computing

A grid is a collection of geographically distributed resources that are independently controlled and administrated by the organizations that owns them. Grid systems provide a set of standards to allow organizations to share these resources on grid applications on an as-needed basis. The term, "the grid", originated from power grids to capturing their similarities in providing electricity on demand. The main objective of grids is to make the computational power available for applications on demand as easily as the electric power grid provides consistent, pervasive, dependable, transparent access to electricity, irrespective of its source, to consumers. Since the mid 1990s, considerable progress has been made towards grid standards and the definition of grid has evolved to accommodate new grid standards. An early definition of the grid given by Ian Forster, widely regarded as one of the 'fathers of the grid', is as follows:

> "A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities." [FOS-1]

The definition of grid was later refined as more research was done on improving grid computing standards. As a complete definition of grids, Ian Forster defines grids in three points.

1) "*coordinates resources that are not subject to centralized control ...*"
2) " *... using standard, open, general-purpose protocols and interfaces"*
3) "*... to deliver nontrivial qualities of service.*" [FOS-02]

The first point empathizes that a grid system consists of a pool of resources that are independently controlled in different administrative domains and makes them available to resource consumers. A distributed computing environment that is subject to centralized control cannot fall in the category of grids. The second point empathizes that a grid is not an application specific system with a set of closed standards. Grid is built using a set of standards, open interfaces and protocols that addresses the fundamental issues including resource discovery, resource access, user authentication and authorization. The third point states that a grid enables controlled and coordinated use of its resources to fulfill users' quality of service demands. These demands include meeting execution deadlines, resource availability, security related issues and availability of resources for simultaneous execution.

2.1.1 Types of Grids

Grids can be categorized by the type of resource that they provide. In general grids often provide more than one type of resource for sharing. Researchers have tried to classify grids in various ways based on their functionality. However, there is no single standard way for characterizations of grids. Based on the intentions of resource sharing, grids can be categorized into the following categories: computational grids, data grids and service grids.

2.1.1.1 Computational Grids

Computational grids provide higher aggregate computational capabilities for resource intensive applications. This is the most commonly used grid type. Resource providers provide computational resources, whose intent is to provide computational power with resource consumers. Depending on how computing capabilities are utilized, computational grids can be

categorized into two forms: distributed supercomputing and high-throughput computing [KRA-1]. Applications requiring distributed supercomputing are generally more complex and highly resource intensive. Grid environments are well suited for these types of applications. On the other hand, applications requiring high-throughput computing consist of streams of incoming jobs [KRA-1].

One example of a computational grid is the TeraGrid which is an open scientific discovery infrastructure [TER-1].

2.1.1.2 Data Grids

Data grids deal with the controlled sharing and management of many distributed data storage devices. Data grids provide specialized infrastructures to applications for storage management and data access. Maintaining high resource capacity within a single organization tends to be costly. Data grids can provide a powerful platform for data mining operations. They can also be cost effective and highly scalable. Some of the well known examples for storage grids are the Amazon S3 [AMA-1] and The European DataGrid Project [WIL-1].

2.1.1.3 Service Grids

Service grids are grids that are service oriented. The main focus of service grids is to provide quality of service guarantees to users. Most common applications that use service grids are on-demand applications, real-time multimedia applications, and collaborative applications. On-demand service grids make specialized services accessible to users on demand. Multimedia grids provide infrastructure for real-time multimedia applications. Grids provide support for

collaborative needs to ensure that continuous real-time interactions between users are established. The flexible, secure and scalable nature of grid environments provides a powerful platform for service oriented grid applications [KRA-1].

2.1.2 Workload of Grids

The workload for grids consists of requests that are made for the execution of grid applications on grid resources. This section discusses two possible request types that the resource consumer can make.

2.1.2.1 On-Demand Requests

On-demand requests are the most common requests made in grid environments. Applications that these requests are made for, are queued to be scheduled on resources. The resource providers do not provide any quality of service guarantees for on-demand requests. The requests are typically served on a best effort basis.

2.1.2.2 Advance reservation Requests

Advance reservation requests are made for applications that begin execution at a time in the future. These requests are used for applications that require QoS guarantees. An advance reservation request includes all the information about the application for the required resources to be reserved. These requests consist of the earliest possible start time of the application, the deadline of the application, which indicates the time that the application must finish its execution, and the execution duration of the application.

2.1.3 Resource Access and Management in Grids

Grids enable the sharing, selection and aggregation of geographically distributed resources for resource consumers. Grid resources are often heterogeneous, independently administrated, controlled by different administrative domains that they belong to, and are accessed through the wide area network. Resource providers have different objectives for sharing resources and their scheduling policies differ from each other. The grid consumer might have to use resources from multiple administrative domains collaboratively. Grid computing has grown larger than what it was originally meant for as applications became larger and more complex. Grid applications require coordinated and controlled access and management of resources in order to meet resource consumer requirements. Consumer satisfaction is measured via the QoS provided by the grid in terms of availability, performance, resource access, and flexibility. This makes resource management in grids a key area of focus.

For a large scale distributed computing environment like a grid system, traditional distributed computing approaches based on centralized policies are not suitable. The integration of hierarchal and decentralized approaches with economical models, are suitable for grid environments [BUY-00]. With these approaches, the grid resource broker mediates the interactions between the resource provider and consumer. The broker has access to information about all available resources. The consumers interact with resource brokers for executing their applications on grid resources. The Broker takes care of all the steps necessary for executing these applications.

The following sub-sections discuss the steps involved in grid resource management.

2.1.3.1 Resource Discovery

In Grids, the first step for selecting a grid resource for application execution is the resource discovery process. Grid resource discovery can be described as a process of matching dynamic and static resource characteristics, to a set of application requirements [IAM-01]. The nature of the grid environment adds many challenges to this process. In grids, resource information and states change dynamically as resource providers introduce new resources or remove resources from being shared or from being available. For applications that require multiple resources to run collaboratively, accurate matching of resource characteristics and applications requirements is important for uninterrupted execution.

In grids, resource information about available resources is stored in Grid Information Services (GIS). As a resource directory service, it maintains/updates the information provided by resource providers and makes that information available upon request [PLA-02]. There have been several research and implementations done on GIS. One such implementation is provided through the Globus Toolkit and it is called the Globus Monitoring and Discovery System (MDS) [GMDS-1]. MDS provides standards for registration and discovery of grid resource statuses and configurations. The resource information data is structured into a decentralized structure that adds scalability. The MDS architecture consists of two main components: Globus Resource Information Service (GRIS) and the Globus Index Information Service (GIIS). The GRIS component provides a uniform means of querying resources to retrieve their current configuration and capabilities. GIIS collects the resource information and provides the collective level indexing and searching functions. GRIS and GIIS interact with each other to keep resource details up-to-date. Other considerable research on GIS is done through the Global Grid Forum

Performance Working Group. Their implementation is called the Grid Monitoring Architecture (GMA). The goal of GMA is to provide a minimal specification that will support required functionality and allow interoperability [GMA-1].

Once resources that match the grid application requirements have been discovered, application jobs are submitted to these resources directly.

2.1.3.2 Advance Reservations

As discussed in section 2.3, the grid workload can be categorized into two different types depending on characteristics of the request for application execution. On-demand requests are most common requests that are satisfied on a best-effort basis. Applications that on-demand requests are made for are submitted to resource(s) immediately after the resource discovery process is complete. Advance reservation requests are made for applications that begin execution at a certain time in the future. Advance reservation requests include the details about the application such as the earliest start time, the deadline of the application, and the estimated execution duration of the application. The advance reservation process is a process for obtaining a resource's capability over a defined time interval through negotiations with the resource provider to ensure the guaranteed availability of the resource [XIN-04]. Usage of advance reservation (AR) provides a powerful mechanism for providing quality of service guarantees to the resource consumers.

The advance reservation concept is used in many other areas such as reservation protocols, RSVP [SCH-98] and ST-II [REI-95], and routing algorithms for networks with

advance reservations [GUE-00]. Usage of advance reservation in grids is first introduced in the Globus Architecture for Reservation and Allocation (GARA) discussed in [FOS2-99]. Advance reservation is used in GARA to provide QoS guarantees for the applications that require simultaneous execution on multiple resources. AR enables the required resources to be reserved in advance and ensures their guaranteed availability. AR has been used in several other contexts such as grid based architectures for dynamic optical networks [LAV-04] and architectures for data-intensive collaboration [FOS-03]. Although AR provides a powerful platform for ensuring QoS guarantees for resource consumers, it may create problems on the resource provider's end such as under-utilization of the resources and maintenance issues due to the dynamic behavior of grids [SID-06]. By holding resources for the reservation time period, AR may harm the flexible resource scheduling that can help to achieve optimized utilization.

2.1.3.3 Matchmaking

Matchmaking in grids is the process of selecting a single resource or a single resource set for a grid application in such a way that minimum application requirements are met and overall system performance is optimized. The main concern in matchmaking is performance and efficiency. A matchmaker can use different criteria for achieving the optimized system performance.

Matchmaking is investigated in several researches. In [RAM-98], the matchmaking framework presented uses a semi-structured data model, classified advertisements (classads), to describe resources. Resources are selected based on their ranking. In [CAS-00], an application level scheduling system is presented where matchmaking is done using different heuristics to

minimize the job completion time. In [SNM-07, BUY2-00, BUY-02, BUY3-00], matchmaking is based on computational economy.

2.1.3.4 Co-allocation

In grids, Co-allocation of resources is required for applications that need simultaneous availability of multiple resources throughout their execution period. One of the examples for applications that need co-allocation of resources can be a data transfer application that uses an optical communication link which consists of multiple channels. Each channel is a resource and the application requires all channels for a complete data transfer. If one or more required resources are not available during the execution, the application cannot start its execution. Reserving the resources in advance is the most effective way to ensure simultaneous availability of multiple resources in multiple domains [FOS2-99]. The main focus of this thesis is co-allocation strategies for advance reservation (AR) requests.

Figure 2-1 shows an example of how co-allocation of resources is done for an application that consists of two parallel tasks (T1 and T2). Both application tasks need to be scheduled simultaneously within the same time window. Resources R1 and R2 are found for T1 considering the type of resources that T1 needs to be scheduled on, and their availabilities. Similarly, resources R3 and R4 are found for T2. In Figure 2-1, the boxes shaded in blue indicate the busy periods for the resources.

From Figure 2-1, resources R2 and R3 are selected for co-allocation because they have idle periods that can allow simultaneous allocation of the application tasks (within t_s and t_e) for a

duration of the application's execution time. Although all resources have idle periods that can allocate the application tasks, simultaneous allocation of resources for both tasks is possible with R2 and R3.

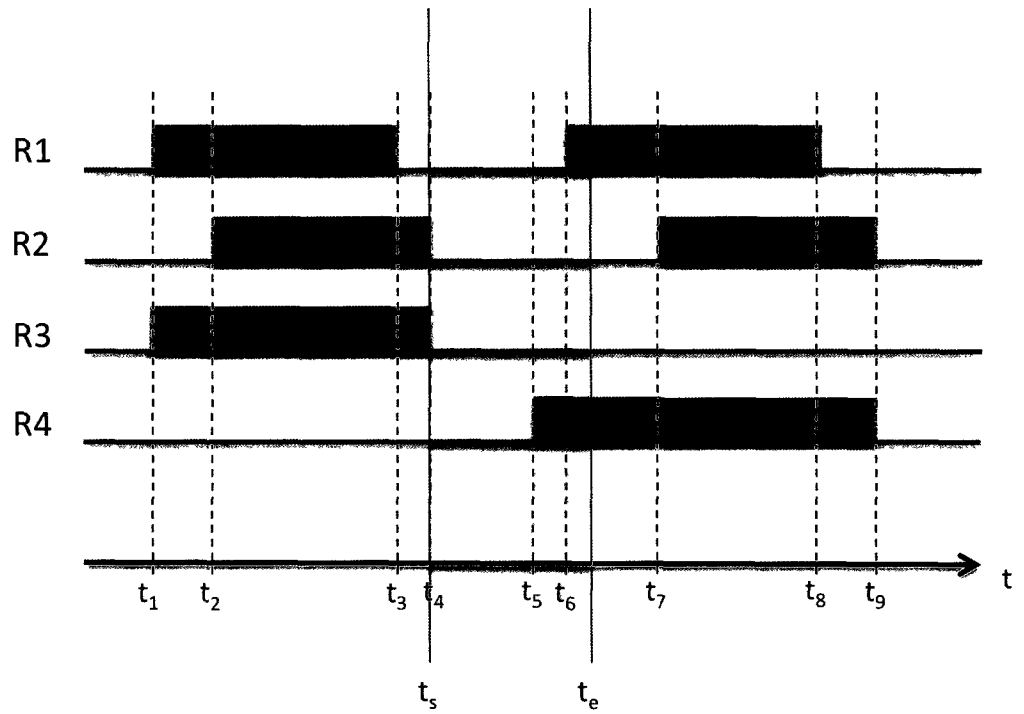

Figure 2-1: An example of resource co-allocation for an application that consists of two parallel tasks.

2.1.4.5 Job Submission and Monitoring

Once the resources are selected for a grid application or advance reservation is made, application tasks are submitted to the resources. A set of grid standards for task submission to resources, monitoring the progress of the task and notifications for successful completion of task execution, makes this process robust and easy for the resource consumers. Job submission and monitoring are challenging tasks in grid environments due to security concerns, the heterogeneous nature of resources and network related concerns [MAT-1]. Some notable

implementations of architectures for job submission are Globus Resource Allocation Manager (GRAM) developed by Globus Alliance [CZA-98] and GridSAM Architecture [GSAM-1]. These systems provide a set of Application Provider Interfaces (API) for job submission and monitoring functionalities to resource consumers. They also provide Service provider Interfaces (SPI), which is configurable by resource providers. Job submission is generally a single command operation or a script that executes a set of commands depending on the API of the platforms used.

Once the job is submitted, progress can be monitored. This is essential for debugging, failure detection and job rescheduling purposes. Monitoring the system should provide current status information about tasks running on resources and provide notifications upon the occurrence of certain events such as system failure and job completion [BAL-03].

2.2 Overview of Cloud Computing

Cloud computing that has a number of similarities with grid computing has started receiving a great deal of attention. A Cloud is a collection of virtualized computing and data storage resources that offers computational and data storage services to consumers based on Service Level Agreements (SLA) [BUY-09]. Based on consumer demands, cloud computing offers readily available services which enable users to increase or decrease the resource capacity they consume [ARM-09]. Consumers can determine the required service level through Quality of Service (QoS) parameters and SLAs. Clouds are available to the consumers in different forms including private and public clouds. Public clouds can be used by all users at large. Notable public cloud environments are available from Google, Amazon, Microsoft Oracle/Sun and

Canonical/Eucalyptus. Amazon's Elastic Compute Cloud (EC2) [AMA-3] and Elastic Block Store (EBS) [AMA-2] are good examples for public cloud environments. Private cloud technologies, where the cloud software is loaded on resources accessed by single group, in an enterprise for example, are available from VMware, Eucalyptus, Citrix and Microsoft.

A large portion of resource management techniques for grids can be extended to the domain of cloud computing. Resource co-allocation algorithms presented in this thesis can be applied to resource management in cloud computing.

2.3 Background on Resource Co-Allocation

This section describes the existing grid frameworks that support co-allocation of resources. It includes a discussion on a representative set of existing works on co-allocation algorithms.

2.3.1 Frameworks that Support Co-Allocation

Several research groups have focused on developing grid middleware systems that support co-allocation of resources. This section presents brief descriptions of some of the notable works.

The Globus Architecture for Reservation and Allocation (GARA) provides a layered architecture that supports resource access and management functionalities, resource discovery, reservation, allocation, co-allocation and co-reservation. It takes advantage of advance reservation to support co-allocation of resources with QoS guarantees to the applications. Usage of AR is first introduced in GARA [FOS-99]. Although GARA provides a framework that supports co-allocation, it does not present any algorithms that can be used to effectively select

resources for co-allocation or determine the best schedule time for the application to start it's execution of the resources selected.

Grid Advance Reservation-based Scheduling Framework (GARS) is an advance reservation-based resource co-allocation framework proposed by National Institute of Advanced Industrial Science and Technology (AIST). The framework uses Web Services resource Framework (WSF) / Grid Security Infrastructure (GSI) and a two-phased commit (2PC) protocol, which supports the generic and secure advance reservation of resources. 2PC protocol allows the reservation process to be completed in a non-blocking manner, simultaneously. This makes GridARS a scalable and more effective co-allocation framework [TAK-07].

Highly-Available Robust Co-scheduler (HARC) is a co-allocation framework that is deployed in several infrastructures such as TeraGrid [TER-1] and UK National Grid Service [UKNG-1]. The framework supports a powerful fault tolerant co-allocation process by taking advantage of its Three-phase Commit Protocol based on the Paxos consensus algorithm [LAM-1]. Co-allocation with HARC is achieved in two steps. First, Resource Manager Components are contacted for timetable information of resources by the clients. Second, based on an analysis of this information, a set of resources are selected and reservation requests are sent to the Acceptor Component of the framework. The framework does not present any algorithms for selecting suitable resources [MAC-11].

2.3.2 Related Work

In this section, co-allocation algorithms and other related works are presented.

In [CAS-09], an efficient online algorithm is presented for co-allocating recourses that provides support for advance reservations. The design of the algorithm is mainly driven by its scalability aspect in large-scale distributed system environments. The algorithm makes the resource search efficient by organizing the temporal availabilities of resources into a two dimensional tree data structure based on their start time and end time. The algorithm first performs a tree-based search to find suitable resources that can start the application at the earliest possible time. If resources are found for all application tasks during the first iterations, the request is accepted. If not, the same step is repeated with a new start time. The new start time is calculated by incrementing the previous start time by a pre-determined value. Searches are repeated until resources are found and the application tasks can be scheduled to execute at the same time, or after the number of iterations has reached a pre-configured max value. The request is rejected if resources are not found for all application tasks.

The algorithm presented in [CAS-09] takes advantage of advance reservation techniques for co-allocating resources. However the algorithm does not provide any QoS guarantees for the co-allocation requests. Instead it tries to reduce the complexity and co-allocate resources at the earliest available time, but do not consider other aspects such as optimizing the resource utilization. Since the algorithm is simplified to ensure the scalability for large-scale distributed systems, it may reject requests that shouldn't be rejected.

In [ZHA-06] a set of adaptive selection algorithms to co-allocate resources is presented. In this study authors utilize a batch scheduler (that uses first-come-first-serve policies) with backfilling support to schedule incoming jobs. Backfilling is used to improve the system

utilization by identifying idle periods in nodes and moving forward smaller jobs that fit into those nodes. There are two different multisite resource selection algorithms proposed – optimal and greedy adaptive resource selection algorithms. For each co-allocation request, both algorithms try to select resource combinations based on the heuristic values calculated for each combination. The heuristic value calculation incorporates network bandwidth and latency between the nodes. The greedy resource selection algorithm employs the greedy heuristic for all tasks in the request. The optimal resource selection algorithm enumerates all the resource combinations for best performance. Performance metrics are calculated based on relative computation costs of the resources.

Batch scheduling techniques typically implement first-come-first-serve (FCFS) policies and are difficult to extend to support advance reservations [ZHA-06]. Pure FCFS policies lead to high fragmentation of resource, low utilization of resources and limited scheduling flexibility. Since there is no support for advance reservation, requests with QoS requirements cannot be served.

In [WAN-03], a co-allocation algorithm which tries to schedule the application tasks on resources at the earliest possible time is presented. The algorithm selects resources for each task, one after the other. The task that can be scheduled the latest, considering the availabilities of the resources it requires, is chosen as the first task. Resources selected for the first task determines the start time of the application. Resources for all other tasks are selected so that the selected resources have the least communication delay, with the resource selected for the first task. The co-allocation request is rejected when required resources for all the tasks are not found.

The algorithm presented in [WAN-03] allows the first task to be the bottleneck task. The algorithm presented in this work, only consider the communication delay with the resource selected for first task, for selecting resources for the other tasks. It also keeps the resource selected for the first task fixed and tries to select resources for other tasks for that particular resource's availability. This might lead to a very inefficient co-allocation of resources.

In [DEC-07], a co-allocation algorithm which is an extension of Heterogeneous-Earliest-Finish-Time algorithm [TOP-02] is presented. The co-allocation process starts with a randomly selected task and the grid resource that gives the lowest earliest-finish-time (EFT) for the task, is selected, and an initial co-allocation window is set. Resources for other tasks are searched to allocate them within the co-allocation window. When two or more resources are found for a particular task, the one that gives the EFT is selected. For a particular task, if none of the resources are available to allocate the task within the current co-allocation window, a new co-allocation window is determined accordingly and all reservations made for the previous tasks are cancelled. The co-allocation process is restarted again for the new co-allocation window. After resources for all the tasks have been selected, start time of the final co-allocation window, is selected as the scheduled time for the application.

The algorithm presented in this paper focuses on ensuring that the co-allocation of resources produces EFT. This approach may not give the best results when considering communication delays, and resource utilization. Requests with deadlines do not require earliest-finish-time as long as their deadline is satisfied. The algorithms presented in this thesis consider other more useful aspects than earliest-finish-time for co-allocation requests.

In [FAR-07], three different co-allocation algorithms are presented. The co-allocation process in all the algorithms starts with determining a set of all possible resource combinations that can accommodate all the application tasks for the period of application's execution time. Co-allocation algorithms presented in [FAR-07] differ between each other in the methods that they use for selecting a resource combination form the set. Co-allocation algorithm, RAN/RAN selects the resource combination randomly. The other co-allocation algorithms select the resource combination based on the relative speeds of the resources in the combination. A cost factor is calculated for each combination using the relative speeds of the resources in the combination. The best combination from the set is determined based on the cost factor values. The resources of the combination selected are reserved at the earliest possible start time. The request is rejected when no resource combinations are found.

Even though the algorithms presented in this thesis can provide an effective co-allocation, they don't scale for large-scale systems. The size of resource combinations increases exponentially with the number of resources.

Chapter 3

Resource Co-Allocation

In grids, resource co-allocation is required in situations where applications require simultaneous availability of multiple resources throughout their execution period. Parallel computing has been one of the essential requirements for high performance grid applications. Some examples of such applications are: data intensive communication using multiple channels, resource intensive multimedia applications, scientific research computations and geo-processing. Such applications need to use multiple resources simultaneously so that component jobs of the applications can execute at the same time. Brokers that support co-allocation of resources need to ensure that different jobs of the Grid application are scheduled to execute at exactly the same time on different resources in potentially different domains. Co-allocation process should consider the resource provider's priority for optimizing system performance while meeting QoS constraints of resource consumers.

The system performance achieved by a broker depends on the co-allocation algorithm used by the broker. The goal of a co-allocation algorithm is to determine the suitable scheduled-time and resources for each co-allocation request in such a way that system performance is optimized. It seeks to minimize work rejected by the system while meeting the QoS and co-allocation constraints of the applications. The most effective way to ensure guaranteed simultaneous availability of multiple resources in multiple domains is to reserve the resources in advance [FOS2-99]. Inefficient selection of resources for co-allocation might lead to performance deterioration in grids. If not managed carefully an AR that needs to hold multiple

resources for the reservation time period, an AR can reduce the flexible resource scheduling that can help to achieve optimized utilization.

This thesis focuses on co-allocation algorithms for advance reservation requests (ARs) with laxity. Laxity of an AR is the difference between deadline of the application and the time at which it would finish executing if it starts executing at its earliest start time. Laxity adds flexibility to the scheduling by allowing more choices for the scheduled time of the application [FAR-07]. Advance reservation is used to ensure the simultaneous availability of resources and meet the QoS constraints of the applications. Once allocated each resource is used by the same application task for the entire duration of the application execution. Such a system is used in the context of ARs by other researches as well. Thesis addresses the problem of co-allocation by dividing the co-allocation process into several steps. There are 16 different co-allocation algorithms presented in this thesis. They use different approaches in two different steps of the co-allocation process, Start task selection and Next task selection. For each step, a different set of sub-algorithms is proposed. Two sets with 4 sub-algorithms in each, makes their 16 combinations to focus on different aspect in co-allocating the resources. Section 3.1, discusses the steps that the co-allocation algorithms follow and presents the two sets of sub-algorithms.

3.1 Resource Co-Allocation Algorithms

All algorithms proposed in this thesis, have a common flow in co-allocating resources for grid applications that consist of concurrent task that need to be scheduled on multiple resources within the same time window. They try to select a resource for each application task, one by one. Resources selected for an application task is influenced by the resources selected for the previous

tasks. The algorithms make sure that resources are selected for all of the application tasks and they can be scheduled to execute in the same time window. There are two key steps in the co-allocation process which play crucial roles in determining the efficiency of the algorithms. One is when selecting the application task to co-allocation process. The other is when selecting resources for the remaining task.

Advance reservation requests made for grid application that require co-allocation, carry all necessary details about the application. The application corresponding to an AR is characterized by the following parameters.

- Earliest Start Time (t_{EST})
    - This indicates the earliest time at which the application can start executing.
- Deadline = (t_D)
    - This indicates the time by which the application needs to complete its execution on the resources.
- Service Time = (t_{ET})
    - This indicates the execution time of the application.

All tasks in a grid application that require co-allocation have the same t_{EST} , t_D and t_{ET}.
Following are the steps that the algorithms follow during the co-allocation process.

Step 1: In this step, algorithms determine resource candidate sets for each application task. The resource candidate set for a task, consists of a set of all the available resources that the task, T_i ,can be allocated for a duration of t_{ET} in the time window, [t_{EST}-t_D]. Each resource in the candidate set of a task, may be able to execute the task at different intervals within t_{EST} and t_D.

Overlap between candidate sets of one or more tasks are handled accordingly during the co-allocation process.

Step 2: In this step, a set of time intervals is determined for each task. An interval set of a task consists of all time intervals on all resources in the candidate set at which the task can be scheduled. Each interval in the set should be able to schedule the task for a duration of t_{ET}. Each resource may contribute multiple intervals within t_{EST} and t_{D}.

Step 3: The next step of the algorithm, is to pick an application task, called the start task, to start the resource selection process. This step is crucial for the algorithm's efficiency. Choosing the appropriate application task to start the co-allocation process is important. There are four sub-algorithms (1A, 1B, 1C and 1D) proposed for selecting the start-task. They follow different approaches and consider different situations for selecting the start task. These sub-algorithms determine the start task as well as the resource and time interval in the resource, for which the task is to be allocated on the resource. The interval selected becomes the allocation window that is used in the following iterations which select resources for the remaining tasks. The start-task is kept same throughout the iterations for selecting resources and intervals for all the remaining tasks.

Step 4: After determining the start-task and the allocation, algorithms try to select resources for all the remaining tasks in several iterations. For each iteration, a resource is selected for one of the remaining tasks in which the task can be scheduled simultaneously. The time interval selected with the start task is used as the initial co-allocation window. Co-allocation window is the time frame in which the application tasks are to be allocated simultaneously. For each remaining task, a resource is selected in such a way that it has an interval with duration greater than or equal to t_{ET} within the current co-allocation time window. Current co-allocation window

is adjusted after each iteration according to the interval selected. Figure 3-1 shows how the co-allocation window is adjusted after two iterations. In the figure, T_1 is the start-task. After intervals for T_2 and T_3 are selected current co-allocation window for T_4's iteration is adjusted according to the intervals selected for T_2 and T_3.

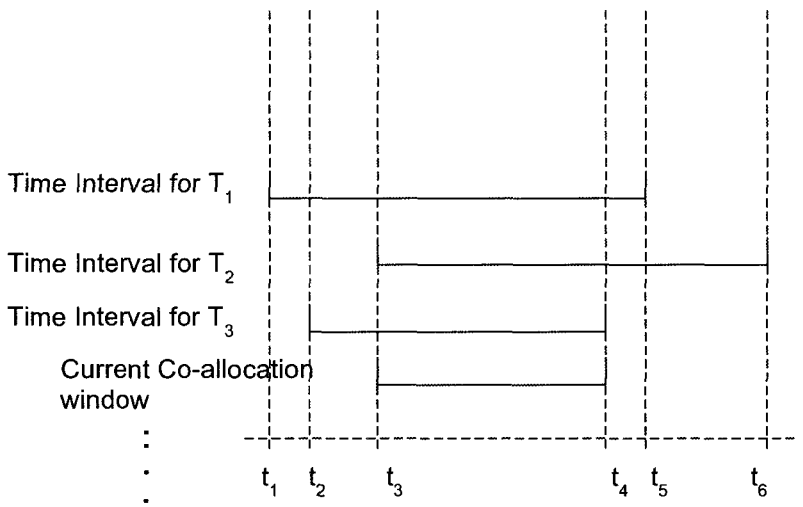

Figure 3-1: An example of how co-allocation window is adjusted

During an iteration, if no time interval can be found to schedule an application task within the current co-allocation window, resources selected for previous tasks are released and the co-allocation process is repeated with the next time interval in the set for start task. Next interval of the start task is selected in accordance with the sub-algorithm used for selecting the starting task. For example, if the sub-algorithm selects a task with largest interval in its interval set as the start task, next largest within the interval set is selected as the next interval. There are four sub-algorithms (1E, 1F, 1G and 1H) proposed for selecting the next-task.

The co-allocation request is rejected when all application tasks cannot be co-allocated for a duration of t_{ET} within t_{EST} and t_D after performing resource selection iterations for all the intervals in start-task's interval set.

If all tasks can be co-allocated in the same time window, resources selected for the tasks are reserved for the duration of t_{ET} and the co-allocation request is accepted. The start time of the final co-allocation window becomes the execution start time of the application.

Following sections discusses the sub-algorithms used for selecting a resource/interval for the application tasks. Table 3-1 lists all the sub-algorithms proposed in this thesis.

3.1.1 Start-Task Selection Sub-Algorithms

Start-Task sub-algorithms play a major role in the co-allocation algorithm. It determines the first application task to start the resource co-allocation process with. The task selected is referred as the start-task and determines the co-allocation time window for the rest of the iteration. These sub-algorithms determine both the resource and interval in the resource for the start-task.

The following sections discuss the sub-algorithms used for selecting the start-task.

3.1.1.1 Sub-Algorithm 1A - First Task with Largest Interval

Sub-Algorithm 1A (First Task with Largest Interval) compares the largest interval in each application task's interval set with other application tasks' largest intervals. It selects a task that has the largest interval than the other application tasks. The selected task becomes the start-task

and the interval and resource that has the interval are also selected. When there is a tie between one or more application tasks, next largest interval is considered to break the tie.

Larger intervals are given priority in the sub-algorithm because it can add more flexibility when choosing the resources and intervals for the remaining tasks.

3.1.1.2 Sub-Algorithm 1B - First Task with Smallest Interval

Sub-Algorithm 1B (First Task with Smallest Interval) compares the smallest interval in each application task's interval set with other application tasks' smallest intervals. It selects a task that has the smallest interval than the other application tasks. The selected task becomes the start-task and the interval and resource that has the interval are also selected. When there is a tie between one or more application tasks, next smallest interval is considered to break the tie.

Smallest interval selection approach taken in this sub-algorithm may lead to a 'best-fit' resource selection for co-allocation while leaving the larger intervals to larger applications.

3.1.1.3 Sub-Algorithm 1C - First Task with Latest Available Interval

Sub-Algorithm 1C (First Task with Latest Available Interval) compares the latest available interval in each application task's interval set with other application tasks' latest available intervals. It selects a task that has the latest interval than the other application tasks. The selected task becomes the start-task and the interval and resource that has the interval are also selected. When there is a tie between one or more application tasks, next latest available interval is considered to break the tie.

The approach taken in this algorithm attempts to push the co-allocation towards the deadline of the application and leave the earliest intervals to the applications with earlier deadlines.

3.1.1.4 Sub-Algorithm 1D - First Task with Earliest Interval

Sub-Algorithm 1D (First Task with Earliest Available Interval) compares the earliest available interval in each application task's interval set with other application tasks' earliest available intervals. It selects a task that has the latest interval than the other application tasks. The selected task becomes the start-task and the interval and resource that has the interval are also selected. When there is a tie between one or more application tasks, next earliest available interval is considered to break the tie.

The approach taken in this algorithm can lead to the application scheduled at the earliest possible time and leave the later intervals to the applications with later deadlines.

3.1.2 Next-Task Selection Sub-Algorithms

These sub-algorithms are used for selecting resources and intervals for the remaining tasks after the start-task selection. Start-task's interval becomes the initial co-allocation window. Co-allocation window is the time slot available for remaining tasks to be allocated within. The resources to be chosen for the remaining tasks need to have interval that can allocate the task within the co-allocation window. The sub-algorithms choose the resources and intervals for the remaining tasks so that simultaneous allocation of resources can be achieved within the co-

allocation window. The next-task selection algorithm plays the role of selecting the appropriate resource and interval for each application task.

Table 3-1: List of Start-Task and Next-Task Selection Sub-algorithms

Sub-algorithm Prefix	Sub-algorithm
Start-Task Selection Sub-algorithms	
1A	First Task with Largest Interval
1B	First Task with Smallest Interval
1C	First Task with Latest Available Interval
1D	First Task with Earliest Available Interval
Next-Task Selection Sub-algorithms	
1E	Resource with Smallest Leftover Time
1F	Resource with Largest Leftover Time
1G	Resource with Highest Utilization
1H	Resource with Lowest Utilization

3.1.2.1 Sub-Algorithm 1E - Resource with Smallest Leftover Time

Sub-Algorithm 1E (Resource with Smallest Leftover Time) selects a resource that produces the smallest leftover time within the co-allocation window. The resource that has this interval is also selected. Leftover time is calculated as:

$$\textit{Leftover time} = \textit{duration of current co-allocation window} - \textit{execution time of the application} \quad \textbf{(3.1)}$$

When there is a tie between one or more resources, next smallest leftover time intervals of the resources are taken into consideration to break the tie.

This approach attempts to fill in smaller intervals while leaving larger intervals for the other applications.

3.1.2.2 Sub-Algorithm 1F - Resource with Largest Leftover Time

Sub-Algorithm 1F (Resource with Largest Leftover Time) selects a resource that produces the largest leftover time within the co-allocation window. The resource that has the interval is also selected. When there is a tie between one or more resources, next largest leftover time intervals of the resources are taken into consideration to break the tie.

This approach attempts to leave a longer leftover space in the interval in which the other applications may be scheduled.

3.1.2.3 Sub-Algorithm 1G - Resource with Highest Utilization

Sub-Algorithm 1G (Resource with Highest Utilization) selects a resource that has the highest utilization than other resources in the time frame: $[t_{EST}, t_D]$. The resource that has this interval is also selected. When there is a tie between one or more resources, a resource is randomly selected.

This approach can lead to a high utilization of the resources used in co-allocation.

3.1.2.4 Sub-Algorithm 1H - Resource with Highest Utilization

Sub-Algorithm 1H (Resource with Highest Utilization) selects a resource that has the lowest utilization than other resources in the time frame: $[t_{EST}, t_D]$. The resource that has this interval is also selected. When there is a tie between one or more resources, a resource is randomly selected.

This approach prioritizes fairness among the resources.

3.2 Simulation Model

3.2.1 Experimental Setup

A simulator was developed to simulate the grid environment, so that the performance of the co-allocation algorithms can be evaluated. It captures different aspects of grid resources, grid applications and their interactions and ensures that a realistic grid environment is modeled. This section discusses the different entities of the simulation model and the workload model used in the simulation.

3.2.1.1 System Architecture

Figure 3-2 shows the system architecture and interactions between the entities. The grid system consists of a collection of resources that reside in multiple administrative domains. All resources register themselves with the Grid Information Service (GIS) entity. GIS in grids, keeps a registry of all the resources registered with it. Grid users submit their requests to the broker. The broker discovers the resources using a GIS. It then chooses appropriate resources that suit the QoS constraints of the resource consumer captured in the co-allocation request and schedule

the application on the resources selected. This system architecture is developed as the simulation environment of this thesis. GridSim ToolKit is used to build the simulator. GridSim ToolKit is discussed in the next section.

3.2.2 GridSim ToolKit

The GridSim toolkit is a Java based library that allows modeling and simulation of entities in parallel and distributed computing environments [BUY1-02]. The toolkit helps to develop a simulation environment with heterogeneous resources, resource brokers. Network configurations and other aspects of grid environments can also be configured using the toolkit. It allows researchers to conduct repeatable and controlled experiments to evaluate the performance of grid resource management and application scheduling algorithms. GridSim toolkit is user-friendly and can be learnt with a few weeks of effort.

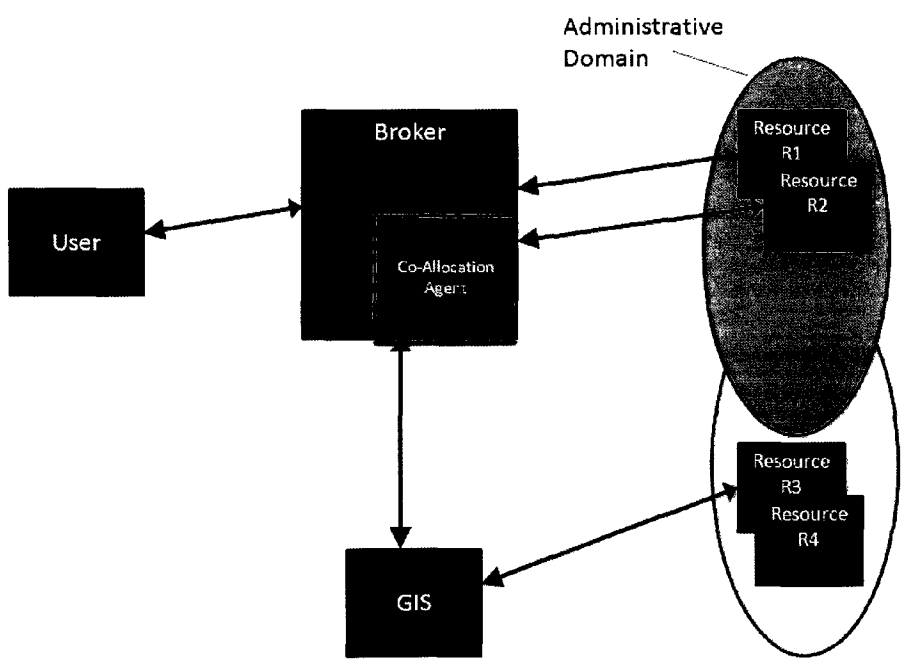

Figure 3-2: System architecture and interactions between the entities.

3.2.3 Simulation Entities

This section describes various grid entities, which make up the simulation environment used in this research. These entities are implemented using the GridSim Toolkit described in section 3.3.2.

Configuration of these entities and implementation of the simulator for the co-allocation algorithms were performed during this Master's thesis research.

3.2.3.1 Source

Source entity is used to simulate the grid users. A single GridSim user entity is used to generate all user requests to the broker. Workload is modeled using the workload parameter values (discussed in section 3.3.3.2). Experimental data collection and computations of performance metrics are also done in the source entity.

3.2.3.2 Broker

Broker entity is the core component of the simulation environment. It is responsible for serving requests from the source. Co-allocation algorithms are executed in the broker. Upon arrival of a user request at the broker, a co-allocation algorithm is executed to determine whether to accept or reject the request. The algorithm also selects the suitable resources and determines the scheduled time for the request.

3.2.3.3 Grid Information Service (GIS)

GIS is responsible for maintaining a registry of the resources in the grid environment. The broker queries the GIS to get the list of available resources. In a grid environment, GIS can

be geographically distributed in order to keep the registry spread across multiple GIS entities. In this research, a single GIS is used and is appropriate for this research that focuses on co-allocation of resources.

3.2.3.4 Resource

In the simulation environment, a number of different types of resources are created. All the resources used in this research have advance reservation capabilities. Application tasks are scheduled on the type of resource that they request for.

Every resource has a resource characteristics component that contains all the records about the resource. The broker determines suitable resources for a user request using the resource characteristics.

A resource interface is used by the broker to access the resource characteristics and for interactions with the resources. This information is used in the co-allocation algorithms.

3.2.3.5 Requests

All requests made by the user in the simulation environment are co-allocation requests. Each request is made by a grid application that consists of a number of parallel tasks, which need to be scheduled within the same time window. In GridSim Toolkit, tasks are represented by gridlets. Thus, each request consists of a number of gridlets, which characterizes the application tasks. The requests also include other details about the applications such as earliest start time, deadline and execution time.

3.2.4 Performance Metrics

This section discusses the performance metrics used in this thesis to evaluate the co-allocation algorithms. Performances of the algorithms are compared using the Blocking Ratio (B), Percentage of Work Rejected (W_R), Utilization (U) and Fairness (Q_R), for various workload and system parameters. The following sub-sections describe the performance metrics in details.

3.2.4.1 Percentage of Work Rejected (W_R)

Percentage of Work Rejected (W_R) measures what portion of the work is rejected by the broker. It is the ratio between sum of the total work of all the requests rejected and the sum of the total work of all the requests submitted. It can be calculated as the following formula.

$$W_R = 100 * (\textit{Sum of the Total Work of the Requests Rejected by the System / Sum of the Total Work of All the Requests Submitted}) \quad \textbf{(3.2)}$$

Total work of each request is calculated by summing their execution times. Since all tasks in a co-allocation request have the same execution window, it can be calculated by multiplying the execution time of the application by the number of application tasks.

3.2.4.2 Blocking Ratio (B)

Blocking Ratio (B) is the ratio between the number of requests rejected by the broker and the number of requests submitted. Requests are rejected by the broker because their deadline cannot be met. It can be calculated as:

$$B = (\textit{Total Number of Requests Rejected} / \textit{Total Number of Requests}) * 100 \quad \textbf{(3.3)}$$

The Total Number of Requests, N, is fixed at 6000 for each simulation run (See table 4-1).

3.2.4.3 Utilization (U)

Utilization (U) measures the fraction that the resources are in use. Average value of the resource utilization is calculated and used in performance comparison. U can be calculated as:

$$U = \textit{Sum of the Busy Units of Time for All the Resources / (Total Observation Time * Total Number of the Resource)} \quad \textbf{(3.4)}$$

3.2.4.4 Fairness (Q_R)

Fairness measures the ability of the system to treat large and small applications equally. Fairness is calculated as:

$$\textit{Fairness } (Q_R) = \textit{(average total work of all requests rejected / average total work of all requests submitted)} \quad \textbf{(3.5)}$$

The metric is used to determine how the algorithms would handle scenarios with different proportions of small and large applications. A value of 1 for Q_R indicates that the algorithm treats the small and large applications equally. A value greater than 1 for Q_R, (the numerator higher than the denominator) indicates that the algorithm is discriminating against the large applications because the average total work for rejected jobs is higher than the average total work for all jobs submitted. A value less than 1 for Q_R, (the numerator lower than the denominator) indicates that the algorithm is discriminating against the small applications over large applications because the average total work for rejected jobs is lower than the average total work for all jobs submitted.

3.2.5 Workload and System parameters

This section describes the workload and system parameters considered in this research.

3.2.5.1 System parameters

This section describes the system parameters that are used to configure the simulation entities.

Number of Resources: This parameter refers to the number of resources in each resource type used in the simulation.

Number of Resource Types: This parameter refers to the number of resource types used in the simulation. Application tasks are scheduled on the resources that match the resource types they require.

3.2.5.2 Workload Parameters

Workload parameters are used to model the workload used in the simulations. In each experiment, one or more of these parameters are varied and the performance of the co-allocation algorithms evaluated.

Number of Parallel tasks: This parameter refers to the number of application tasks in each request. This parameter is generated by using a uniform distribution and the default value of this parameter is generated by using uniform distribution and the default values for lower and upper bounds are 2 and 6 respectively.

Arrival Rate (λ): Arrival rate is the rate at which requests arrive on the system. The Poisson arrival process is used to characterize the arrival of requests in this research. The Poisson arrival process is used in several other research works (see [FAR-03] and [RCP-01] for example). The

parameter, λ, is the number of arrivals per second. The default value of λ used in this thesis is 0.4 arrivals per second.

Service Time: This parameter refers to the execution time of an application corresponding to a request. In a co-allocation request, all application tasks have the same execution time. The service time is assumed to be uniformly distributed and the default values of the lower bound and the upper bound are 10 minutes and 90 minutes respectively. A similar modeling of service time is used in several other researches (see [MAJ-09] and [SUL- 04] for example).

Earliest Start Time: This parameter refers to the earliest possible time that the application can be executed on the resources. This is modeled using a delay parameter, which is uniformly distributed with a lower bound of 0 hours and an upper bound of 10 hours. The earliest start time of the request is calculated by adding the value of the delay parameter with the arrival time of the request. This method ensures more realistic modeling of the earliest start times for the user requests.

Laxity: Laxity of an AR request is defined as the ratio of the difference between the deadline and the earliest start time, and its execution time:

$$Laxity = (Deadline - Earliest\ start\ time) / Execution\ time \quad \textbf{(3.6)}$$

Laxity influences the deadline of the application. The following equation is used to calculate the deadline of the application.

$$Deadline = Earliest\ start\ time + (Laxity * Execution\ time) \quad \textbf{(3.7)}$$

3.2.6 Verification of Simulation Results

This section discusses the various ways used to verify the validity of the simulation runs performed in this thesis.

3.2.6.1 Manual Verification

To validate the simulation runs performed in this research, some verifications are performed manually with the aid of debugging and reporting functionalities of the programming environment. Simulation results are compared with manually calculated results using interim results collected throughout the simulation run. Several states of the simulation environment are also recorded and verified to ensure correctness of the simulation model.

3.2.6.2 Little's Law

Little's Law [LIT-61] was also used for verifications of the simulation experiments. Little's Law states that the mean number of requests in a system is equal to the product of the mean response time and the system throughput. Values of these parameters were recorded during simulation runs and the relationship among them was verified using little's Law.

Chapter 4

Performance Analysis

This chapter discusses the results of various simulation experiments conducted for evaluating the performance of the co-allocation algorithms proposed in this thesis. The simulation model developed (see Chapter 3) to simulate the grid environment to capture different aspects of grid resources, grid applications and their interactions. It also provides a controlled environment to conduct experiments that can be used to effectively study the performance of the co-allocation algorithms.

The simulation model that was used for the experiments, has been discussed in detail in Section 3.3.1 while the workload model used in the experiments, has been discussed in Section 3.3.3. To evaluate the performance of the algorithms, a factor at a time approach is used. In each simulation experiment, all workload and system parameters are held at their default values while varying the parameter (factor) of interest. Varying a single parameter helps identify the impact of that parameter on the performance of the algorithms. As discussed in Section 3.3.3, default values for these parameters, were chosen carefully to ensure that the simulation setup can effectively evaluate the performance of the co-allocation algorithms. Table 1 shows the list of system and workload parameters and their default values used in the experiments.

4.1 Objectives

The objectives of this chapter are to compare the co-allocation algorithms proposed in this thesis. As discussed in earlier sections, the algorithms are combinations of two sets of sub-algorithms. Each sub-algorithm focuses on a different aspect and can impact the performance of

the algorithm. Experiments were conducted to analyze the effect of various workload and system parameters that can examine the performance of the algorithms. Each simulation run considered 6000 request arrivals. The simulation run for a given combination of system and workload parameters was repeated multiple times. Each performance metric reported is an average computed over these multiple runs.

Table 4-1: Default Values of System and Workload parameters

Parameter	**Value**
Workload parameters	
Arrival Rate, λ	0.4/ mins
Total Number of Requests, N	5000
Number of concurrent tasks, I	U(2,6)
Difference between request arrival time and t_{EST}, D_{EST}	U(0,10) hours
Execution time, t_{ET}	U(10, 90) mins
Laxity, L	2
System Parameters	
Number of Resource Types, J	6
Number of resources in each type, M_J	15

4.2 Impact of Co-Allocation Algorithms on Performance

In order to study the performance of the co-allocation algorithms, workloads with co-allocation advance reservation requests is used. Following sections discuss the results of experiments conducted and the impact that a given workload parameter has on the performance of the algorithms. Performance metrics used in these experiments have been discussed in Section 3.3.2.

4.2.1 Comparison of the Sub-Algorithms

This section studies the performance of co-allocation algorithms proposed in this thesis. As discussed in earlier sections, there are 16 different algorithms derived in this thesis by using different approaches in the two different steps in the co-allocation process. Two sets with 4 sub-algorithms in each are proposed (see Section 3.1). Each sub-algorithm in a set contributes differently to the efficiency of the algorithm combination. Each sub-algorithm in a set is compared with others in the set while keeping the sub-algorithm from the other set the same in the pair: start-task selection and next-task selection sub-algorithms. For example, to evaluate the best and the worst start-task selection sub-algorithms, their combinations with the same next-task selection algorithm, are used. In this way, the sub-algorithm can be evaluated against others to choose the best and the worst within the set. This section studies the effectiveness of the sub-algorithms and the combinations.

All user requests used in the experiments are co-allocation advance reservation requests with laxity. Laxity allows flexibility to the allocation of resources to the grid applications. As discussed in section 3.3.3, arrival of the requests, are modeled as a Poisson arrival process with arrival rate λ. Each request is made for a grid application that requires multiple resources to be allocated simultaneously during its execution period. Application tasks may require different types of resources. Once the request arrives at the broker end, the broker tries to select and allocate resources for the application. Co-allocation algorithm plays the role of selecting appropriate resources for the grid application.

4.2.1.1 Performance Comparisons of Start-task Selection Sub-Algorithms

Figures 4.1 – 4.6 show the impact of varying arrival rate for the set of sub-algorithms responsible for selecting the start-task in the algorithm combination. In the graphs shown, sub-algorithm 1E (Resource with Smallest Leftover Time) is used as the next-task selection sub-algorithm. During the experiment, the arrival rate, λ, is varied between 0.05 and 1 requests per min. Figure 4.1 shows the impact of arrival rate on Blocking Ratio. Each line in the graph corresponds to a combination of sub-algorithms, where each combination consists of a start-task selection sub-algorithm and the next-task selection sub-algorithm, 1E (Resource with Smallest Leftover Time). Figure 4.1 shows that as the arrival rate increases, B increases for all the sub-algorithms 1A-1D. As the arrival rate increases, the competition for resources increases and the rate of rejection increases. For low and high arrival rates, differences among these first-task selection sub-algorithms are not visible. This is because when λ is at very low values, the competition for resources is low and all the four sub-algorithms are able to accept most requests. Similarly, for very high λ values, the resources tend to saturate reducing the difference in performance of the sub-algorithms. However, at the intermediate arrival rates the difference in performance of sub-algorithms is significant. For a better understanding, intermediate arrivals are further illustrated in Figure 4-2 that presents an expanded version of a part of Figure 4-1 that corresponds to a range of intermediate arrival rates. Between the sub-algorithms 1C (First Task with Latest Available Interval) and 1D (First Task with Earliest Available Interval), 1C (First Task with Latest Available Interval) showed better performance. Pushing the resource allocation towards the deadline, tends to give rise to a better performance than choosing the earliest available schedule time for the application. Between 1B (First Task with Smallest Interval) and 1A (First Task with Largest Interval), 1B demonstrates a better performance. Choosing a

resource with a smaller free interval and leaving the resources with larger free intervals for future allocations tends to reduce resource fragmentation and improves performance. In the start-task selection algorithms set, 1C (First Task with Latest Available Interval) and 1A (First Task with Largest Interval) showed the best and the worst performances respectively in terms of B, than the others. Selecting a resource that pushes the resource allocation towards the deadline seems to give rise to the best system performance by leaving the resources with earlier intervals free for future allocations. Selecting a resource with largest available interval leads to the worst performance by using up the larger free intervals that could have been used for future resource allocations.

Figure 4-3 shows the impact of arrival rate on Utilization. Utilization (U) indicates how the resources are utilized. Higher utilization indicates better performance as resource providers wants higher utilization of the resources they provide. For all arrival rates, difference in utilization is clearly visible. In comparison using U, 1C (First Task with Latest Available Interval) and 1D (First Task with Earliest Available Interval) showed the best and the worst performances respectively. Selecting a resource that can push the allocation farther towards the deadline, leaves the resources with earlier intervals free for future allocations. This improves resource utilization. 1A (First Task with Largest Interval) shows the second worst performance in terms of resource utilization among the start-task selection algorithms.

Figures 4-4 and 4-5 shows the impact of arrival rate on percentage of the work rejected by the broker (W_R). W_R indicates how much of the work submitted rejected by the broker. Lesser W_R indicates better performance. Similar to B, as the arrival rate increases, W_R increases. For

lower and higher arrival rates, differences in performances of the first-task selection sub-algorithms are small. As intermediate arrival rates the sub-algorithms perform differently from one another. Similar to the comparisons using B, 1C (First Task with Latest Available Interval) and 1A (First Task with Largest Interval) shows best and worst performances respectively. By pushing the resource co-allocation towards the deadline (1C) of the application, more workload is accepted by the broker. Choosing a resource with the largest available free interval for the start-task (1A) tends to give rise to a poorer system performance. This is because using up larger free intervals for the tasks, hurts the future allocation of resources to larger applications.

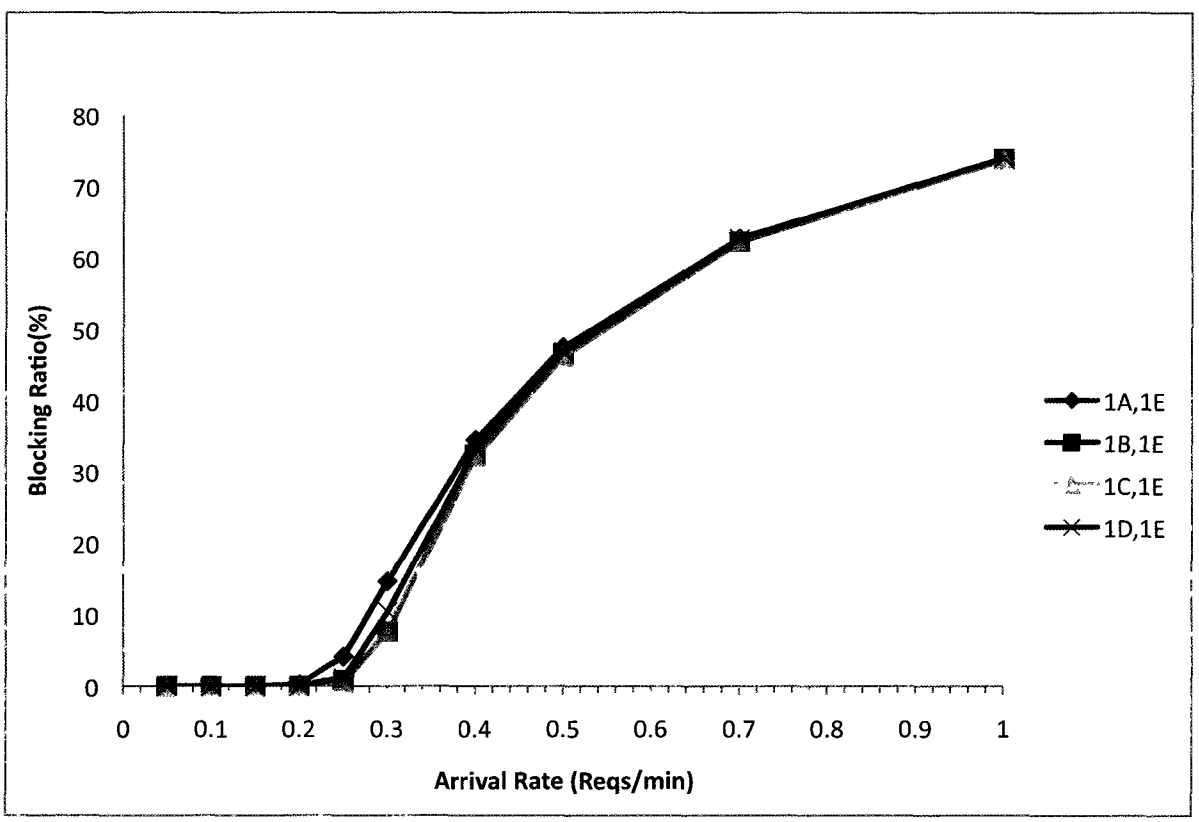

Figure 4-1: Comparison of sub-algorithms using the impact of arrival rate on Blocking Ratio (B)

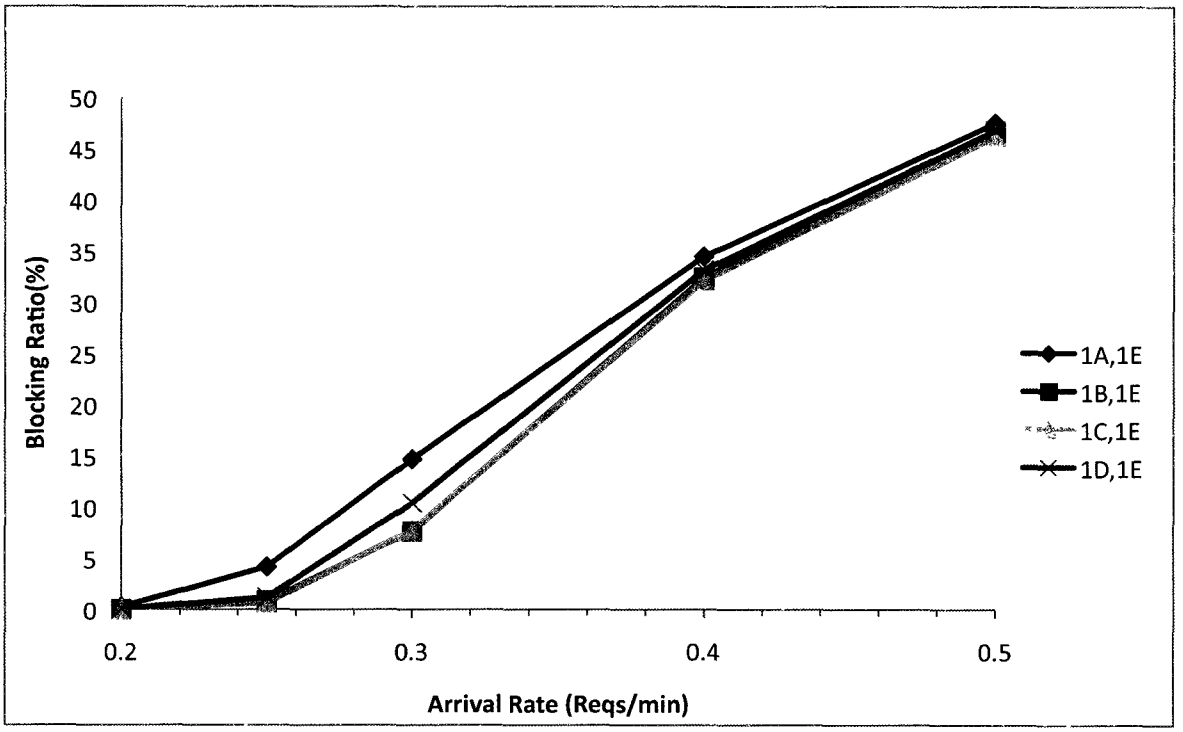

Figure 4-2: Expanded Version of Part of Figure 4-1 Corresponding to Intermediate Arrival Rates

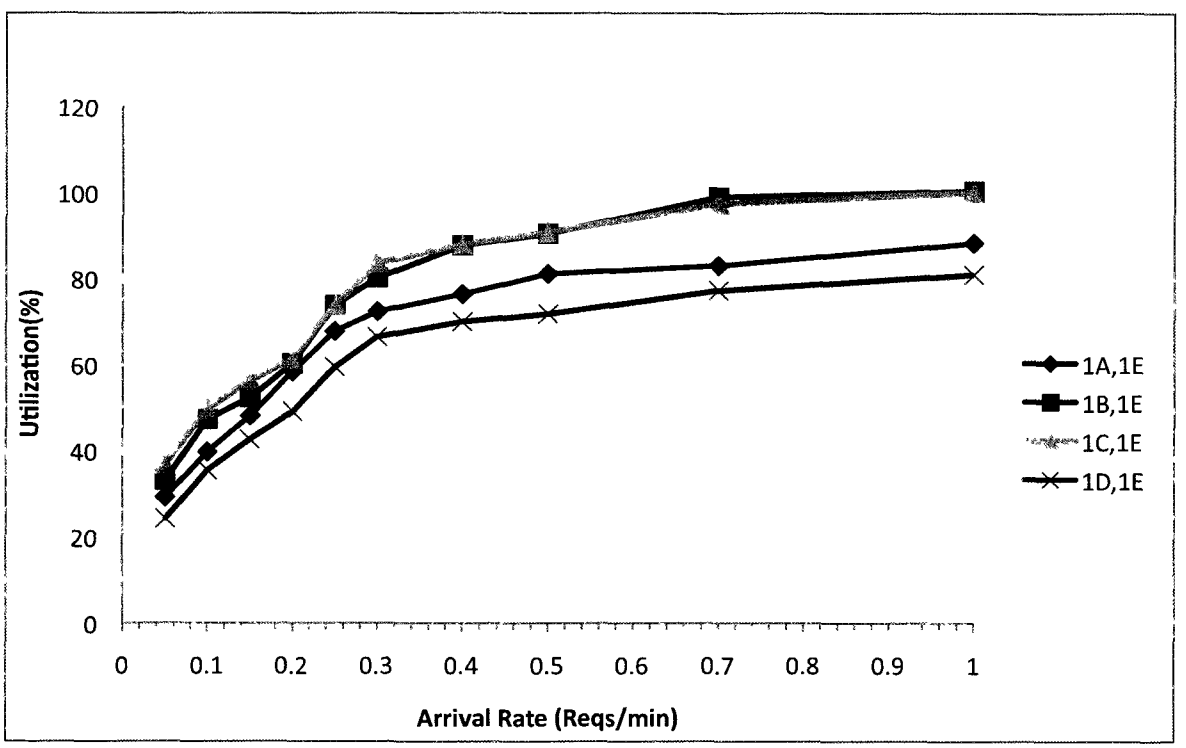

Figure 4-3: Comparison of sub-algorithms using the impact of arrival rate on Utilization (U)

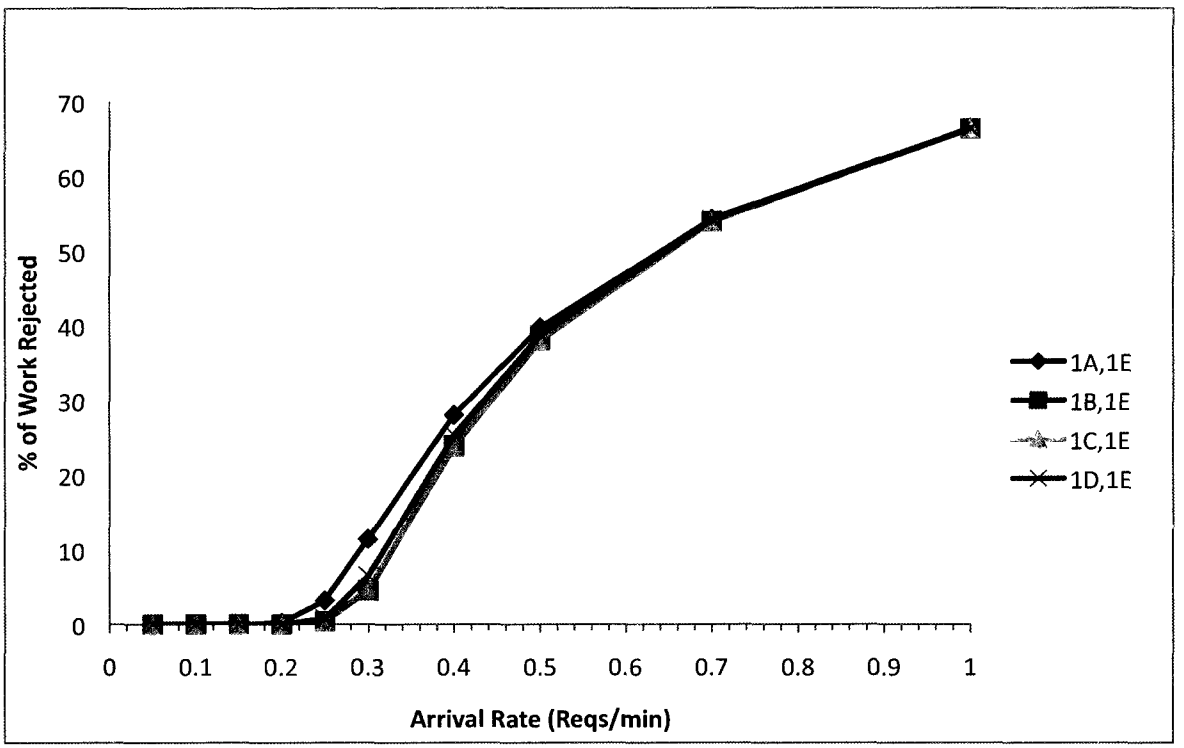

Figure 4-4: Comparison of sub-algorithms using the impact of arrival rate on Work Rejected (WR)

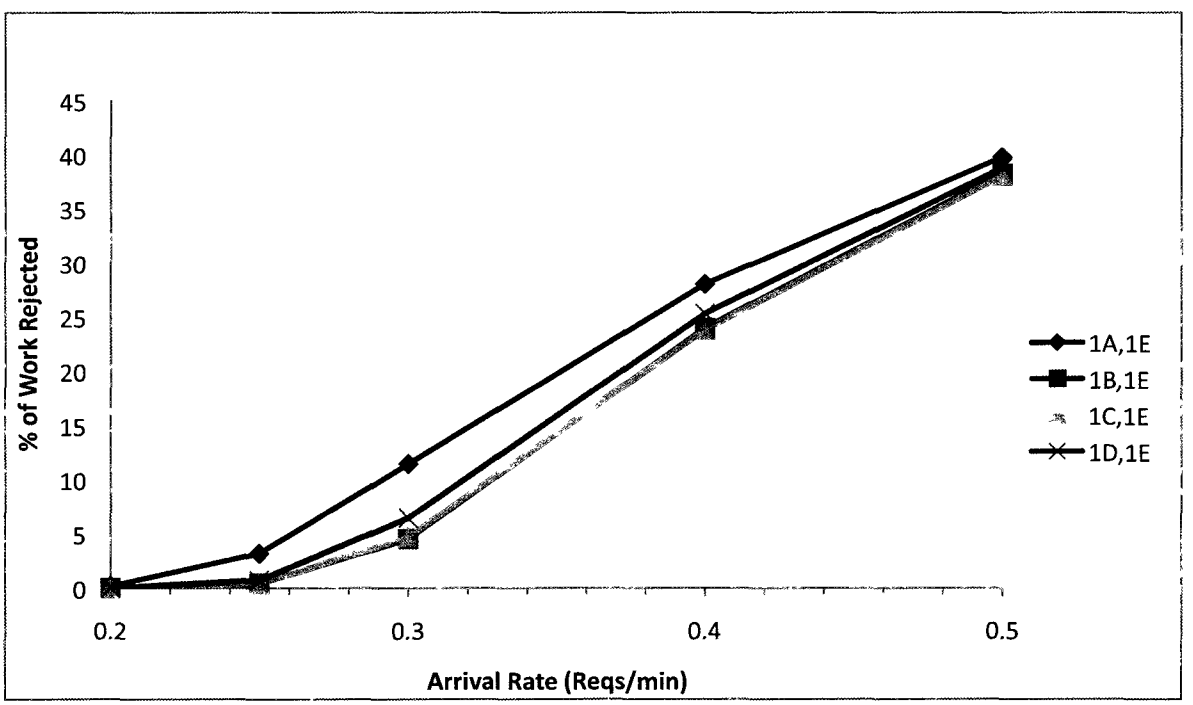

Figure 4-5: Expanded Version of Part of Figure 4-4 Corresponding to Intermediate Arrival Rates

Figure 4.6 shows the impact of arrival rate on fairness for the requests (Q_R). Q_R indicates how fairly large and small request are treated by the broker. Q_R closer to 1 indicates a better performance. When Q_R is higher than 1, larger requests are accepted more than smaller requests. Similarly when Q_R is lower than 1, smaller requests are accepted more than larger requests. Ability of the algorithm's to treat small and large grid applications equally determines the effectiveness of the algorithm. Sub-algorithm 1A (First Task with Largest Interval) demonstrates the best performance for most arrival rates in terms of Q_R. At very low arrival rates, when competition for resources is low, all the sub-algorithms demonstrate a similar fairness. Also at high arrival rates at which the resources tend to get saturated, all the sub-algorithms demonstrate a comparable fairness.

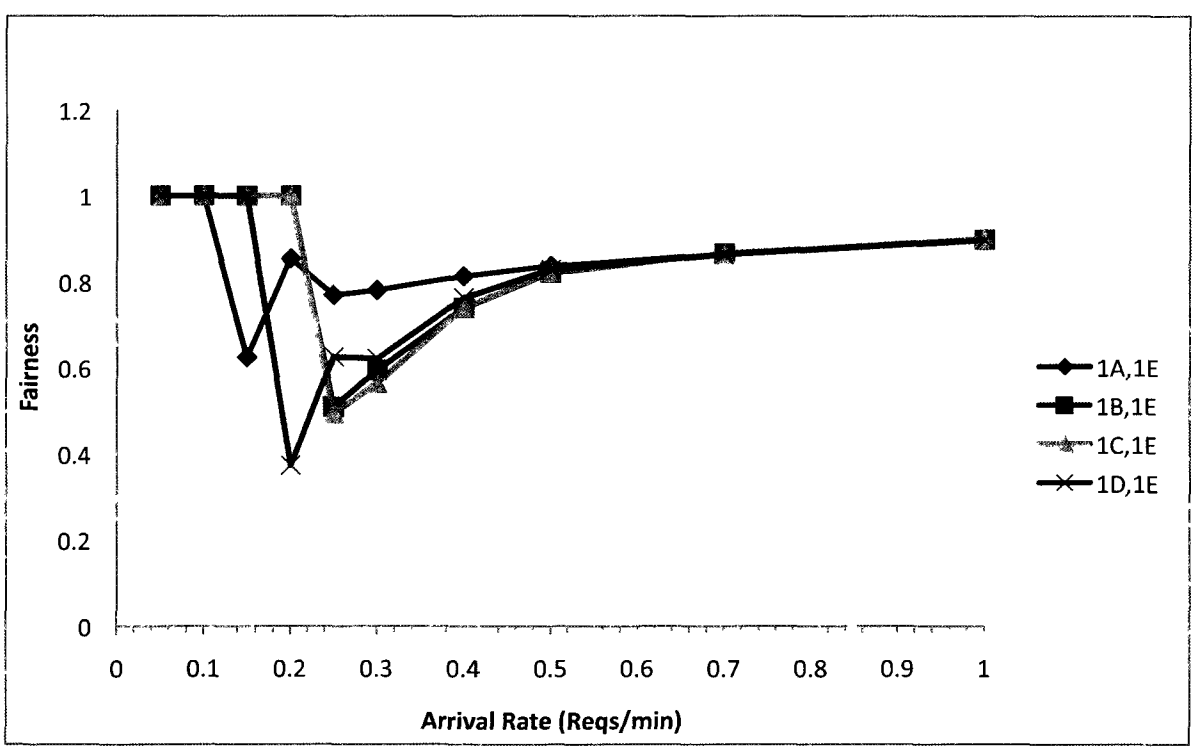

Figure 4-6: Comparison of sub-algorithms using the impact of arrival rate on Fairness (Q_R)

In Figures 4-2 and 4-5, the differences in performance of the sub-algorithms are larger for a range of intermediate arrival rates. Results presented in all these figures, indicate that 1C (First Task with Latest Available Interval) and 1A (First Task with Largest Interval) demonstrates the best and the worst performance respectively.

Figure 4.6 demonstrates a non-monotonic behavior for all the sub-algorithm combinations displayed in the figure. At very low arrival rates none of the requests are rejected, leading to a fair system ($Q_R = 1$). It is interesting to note that for a range of intermediate values of arrival rates $Q_R < 1$ is achieved. This implies the rejection of a higher number of small jobs at these arrival rates. Q_R is observed to improve at higher arrival rates.

4.2.1.2 Performance Comparisons of Next-task Selection Sub-Algorithms

Figures 4-7 – 4-10 show the impact of varying arrival rate on performance achieved by the set of next-task selection sub-algorithms in the algorithm combination. In the graphs shown the start-task selection sub-algorithm is kept same, to 1C (First Task with Latest Available Interval). Start-task, 1C (First Task with Latest Available Interval), is chosen because it is the best sub-algorithm in its set and in terms of B, combinations with 1C (First Task with Latest Available Interval) shows more visible performance differences than combinations with other first-task sub-algorithms. Next-task sub-algorithms are compared with each other using all the four performance metrics.

As observed with Figures 4-1 – 4.6, the next-start selection sub-algorithms perform comparably at low and high arrival rates. In Figure 4-7 and 4-10, intermediate arrival rates result

in differences in performance of the sub-algorithms. Results indicate that 1G (Resource with Highest Utilization) and 1H (Resource with Highest Utilization) demonstrates the best and the worst performance respectively.

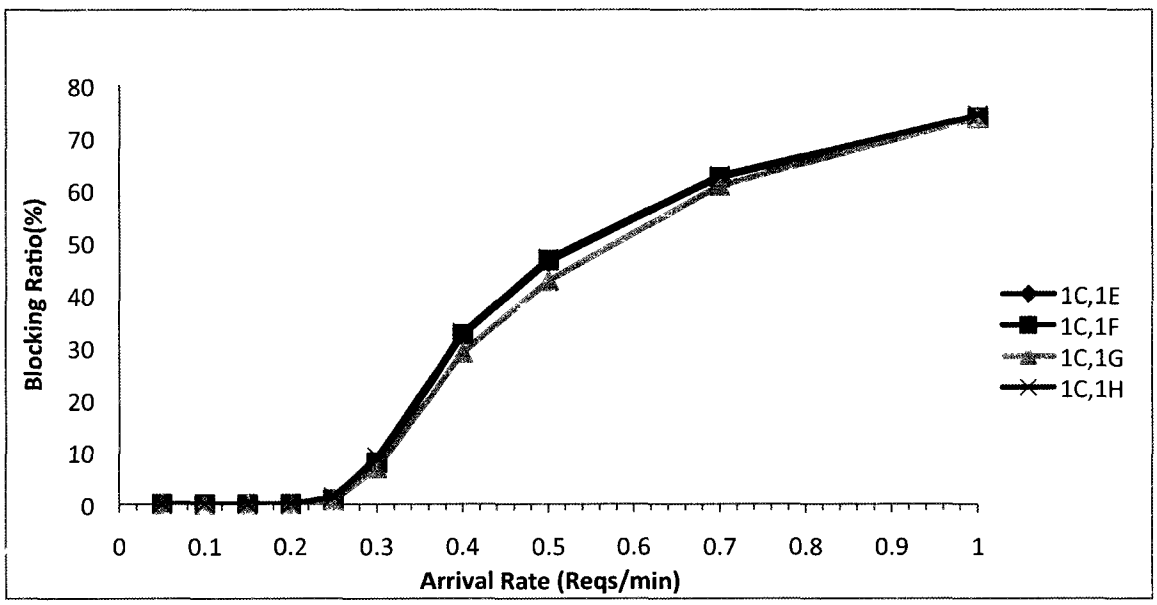

Figure 4-7: Comparison of sub-algorithms using the impact of arrival rate on Blocking Ratio (B)

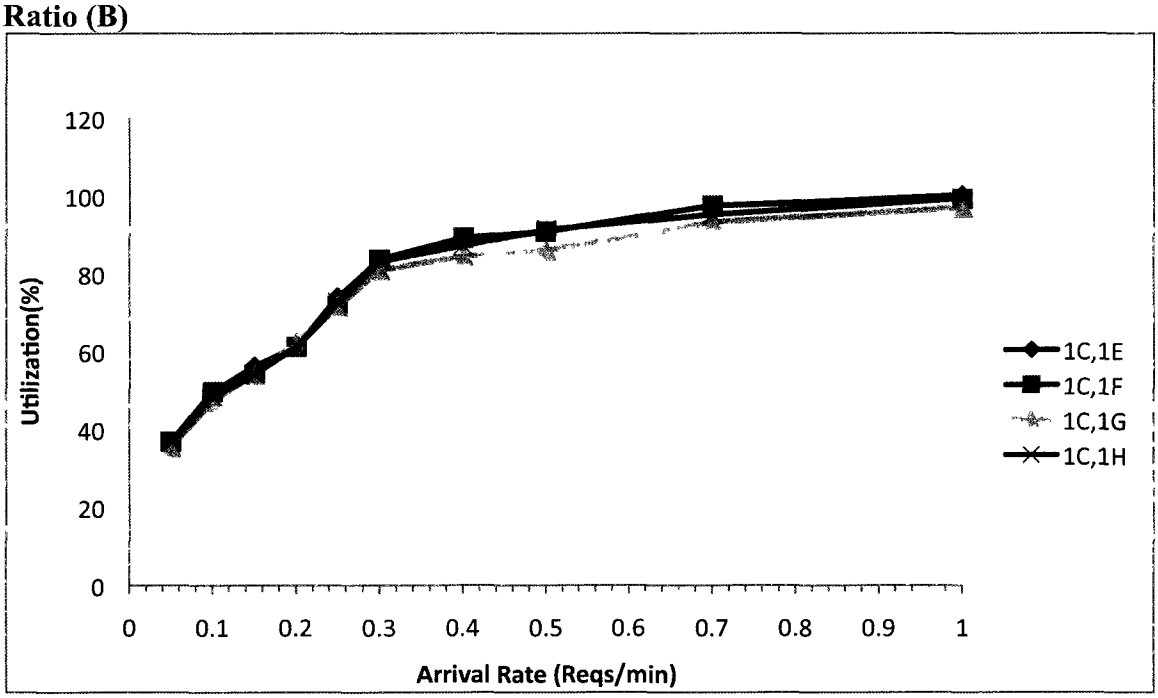

Figure 4-8: Comparison of sub-algorithms using the impact of arrival rate on Utilization (U)

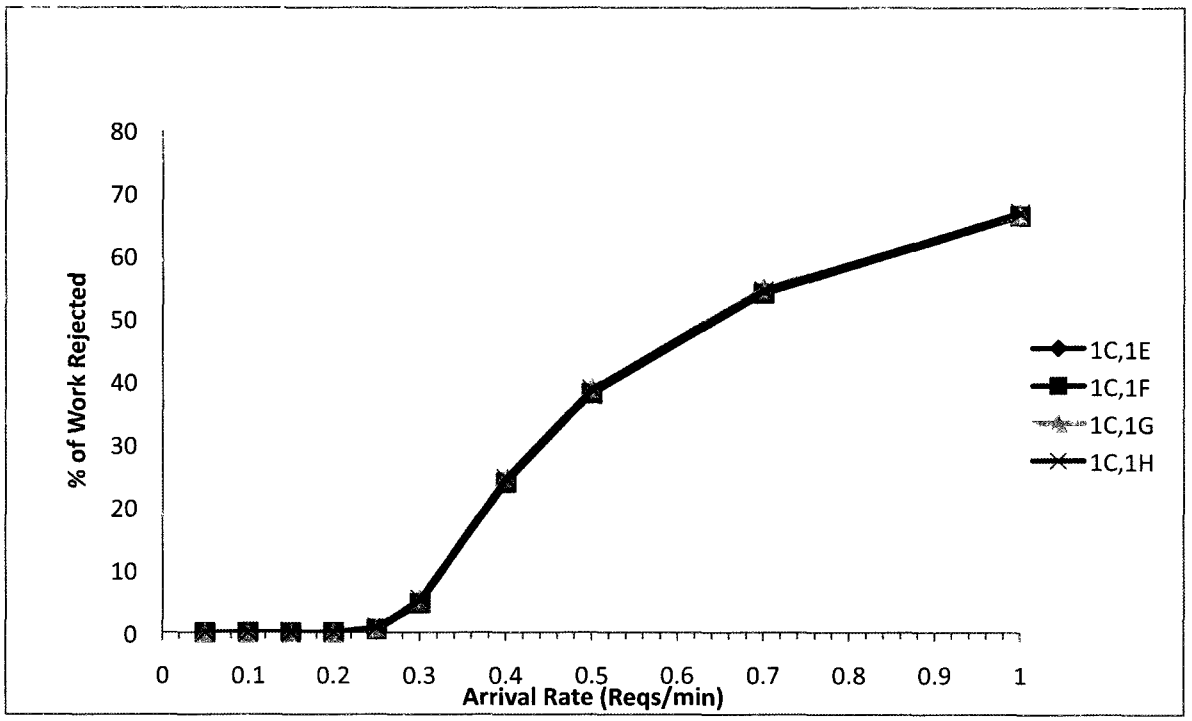

Figure 4-9: Comparison of sub-algorithms using the impact of arrival rate on Work Rejected (WR)

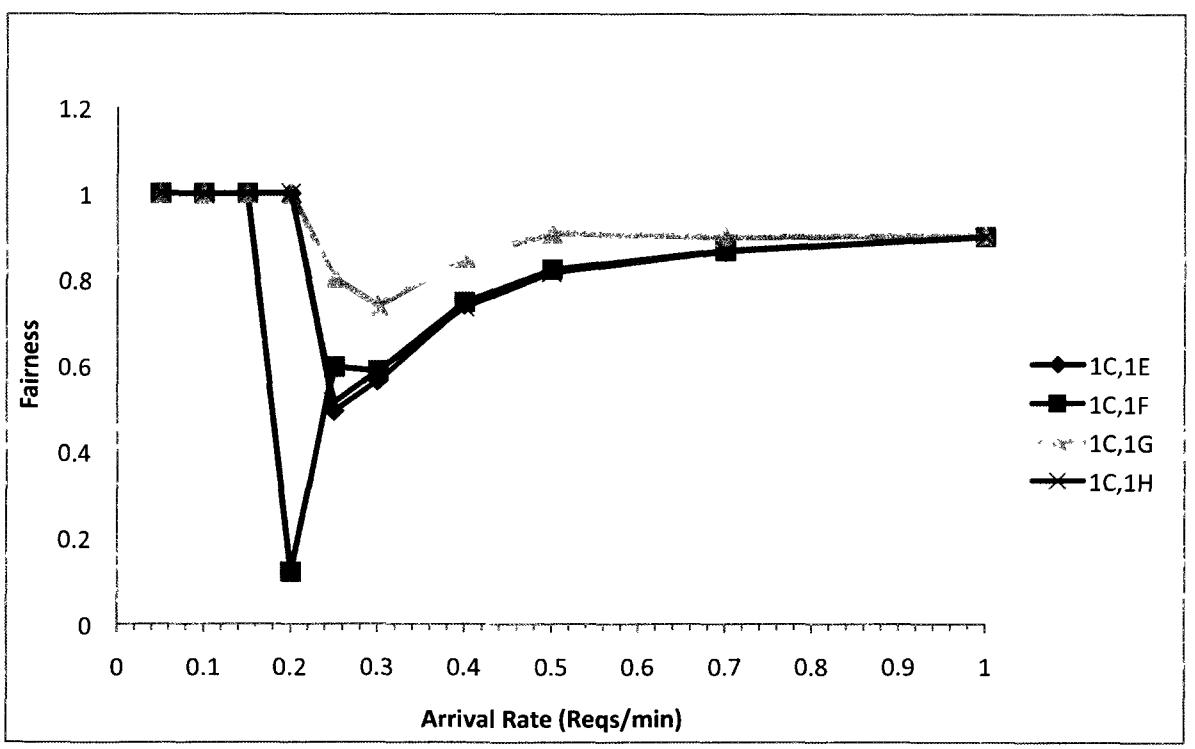

Figure 4-10: Comparison of sub-algorithms using the impact of arrival rate on Fairness (Q_R)

B is an important parameter because it reflects the ability of the system to accept requests and has a direct impact on the revenue earned by the resource provider on a market oriented grid. Thus, B is chosen as the performance metric to be used for determining the best and the worst sub-algorithms. From B values observed for the sub-algorithms, it is concluded that for the first-task selection sub-algorithm set, 1C (First Task with Latest Available Interval) and 1A (First Task with Largest Interval) showed the best and worst performance respectively. Similarly, in the next-task selection sub-algorithm set, 1G (Resource with Highest Utilization) and 1H (Resource with Highest Utilization) showed the best and the worst performance respectively. From further analysis, it was found that the algorithm combination that includes both best sub-algorithms achieves the best performance. Similarly the algorithm combination that includes both worst sub-algorithms achieves the worst performance. The results of performance comparisons for various workload parameters are presented in Table A.1 – Table A.10 included in the Appendix. As can be clearly seen in majority of the cases (each case captured in a specific row of a given table) 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) gives rise to lowest B whereas 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) produces the highest B. Thus, 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) is referred to as the best and 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) as the worst performer.

The experimental results indicate that using the latest available interval in choosing the resources for the first task (1C) and using the highest utilized resource in choosing the resources for the remaining tasks (1G) leads to a high system performance. Choosing the best and the worst combination allowed us to focus only on two combinations rather than 16 different

combinations in the remaining experiments. Impact of varying other workload and system parameters on system performance achieved by the best and the worst combinations wraps the behavior of all 16 combinations.

4.2.1.3 Effects of Arrival Rate on Co-allocation Algorithms

Figures 4-11 – 4-14, shows the impacts of varying arrival rates on best and worse algorithm combinations, 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) and 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization). In the graphs, the difference in performance of the algorithms, are clearly visible for intermediate arrival rates because the performance shows the accumulated effectiveness of the best and the worst sub-algorithm choices.

In Figure 4-11, as expected, values observed for intermediate arrival rates show a significant difference in performance. For lower and higher arrival rates difference is not visible.

In figure 4-12, values observed for U shows higher difference for higher arrival rates. This indicates that the 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) combination becomes more effective when the competition for resource is really high. This is because sub-algorithm IG tries to choose highly utilized resources for application tasks while 1H (Resource with Highest Utilization) tries to choose lowly utilized resources for the application tasks. Allocating tasks to highly utilized resources first and leaving the resources with lower utilizations for future allocations seems to give rise to a higher performance. The gap between utilizations achieved by the combinations tends to widen as arrival rate increases

leading to an increase in competition for resources. 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) outperforms 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) for intermediate arrival rates in terms of both W_R and Q_R (See Figures 4-13 and 4-14). In Figure 4-14, it is observed that for most arrival rates, Q_R values for 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) Combination are closer to 1 compared to Q_R values for 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) Combination. A Q_R values closer to 1 indicates that the algorithm combination is treating large and small applications fairly, thus the combination showed better performance.

4.2.2 Effect of Laxity

This section focuses on determining the effect of the request's laxity on performance. In this experiment laxity values are varied while keeping the other parameters at their default values. Deadline of a request is determined by using Equation 3.6 in Section 3.3.5.2.5. Varying L helps to evaluate the impact the amount of this flexibility in the requests, has on the system performance of the algorithm combinations. In this experiment, L is varied between 1 and 5. Higher the laxity the higher is flexibility in scheduling the request.

As expected, in Figure 4-15, as L increases B is decreases. This is because an increase in the flexibility associated with a request increases its probability of being schedulable. The gap between the B values of combinations 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) and 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) widens as L increases. Allowing more flexibility in scheduling resources for

application tasks lets 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) to select lower utilized resources. This leads to poorer resource selection when using 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) as L increases. However 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) combination handles the flexibility more effectively and shows better performance for larger L.

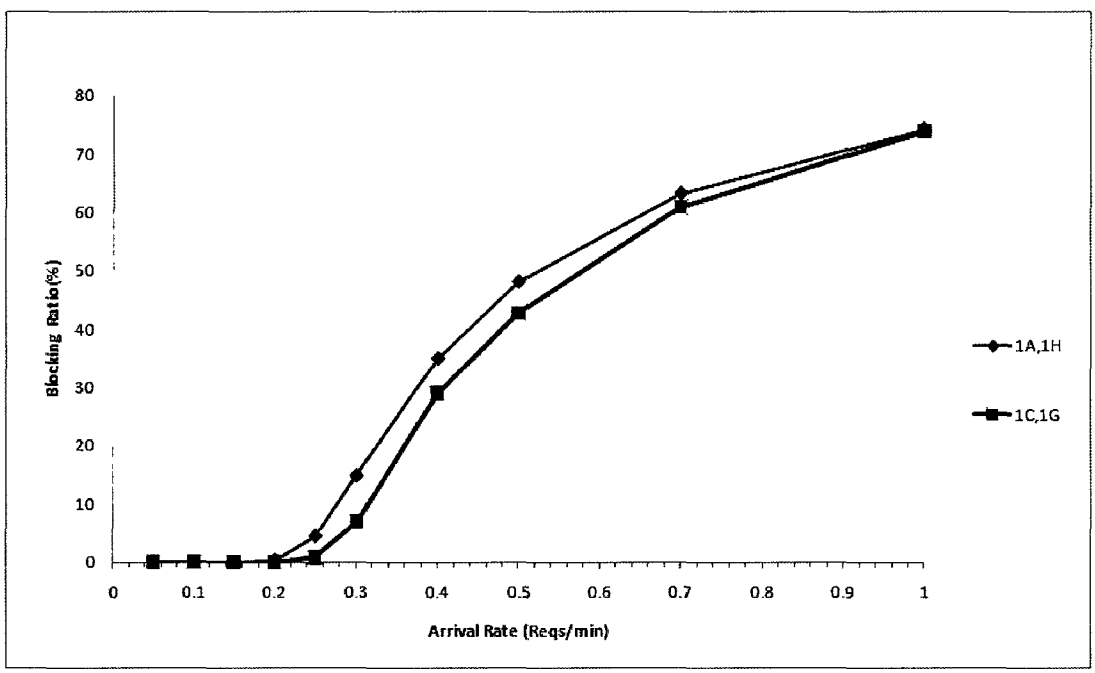

Figure 4-11: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of arrival rate on Blocking Ratio (B)

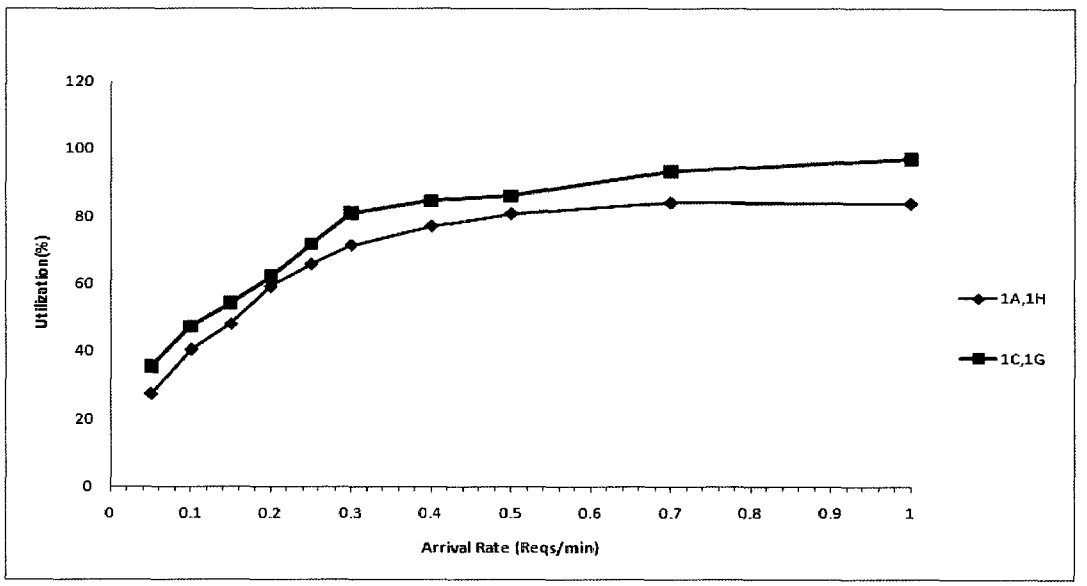

Figure 4-12: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of arrival rate on Utilization (U)

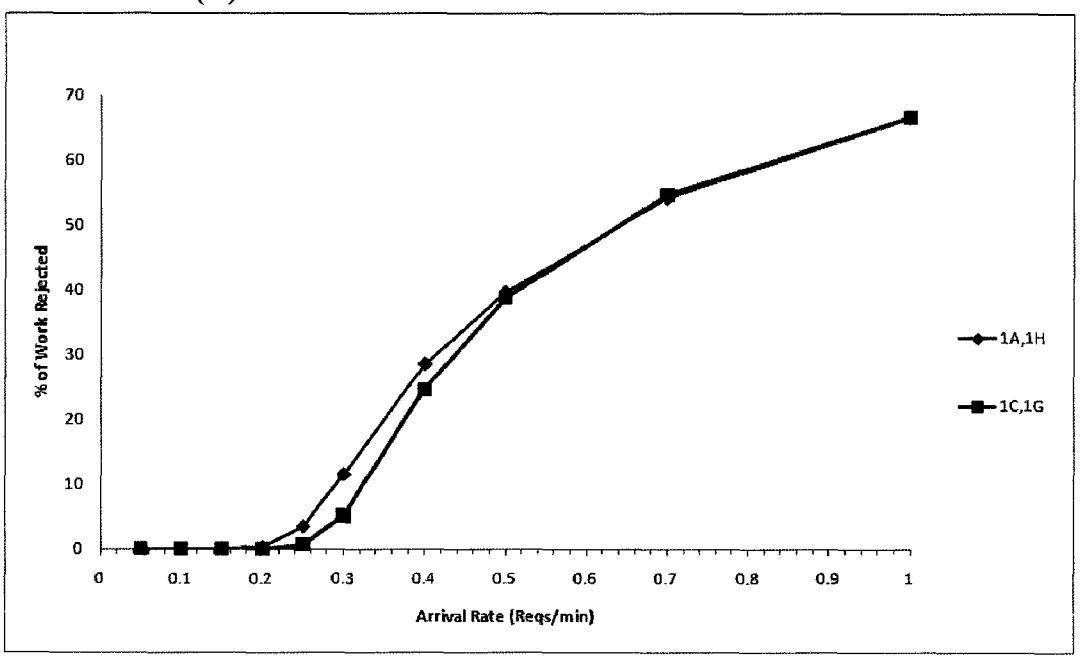

Figure 4-13: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of arrival rate on Percentage of Work rejected (W_R)

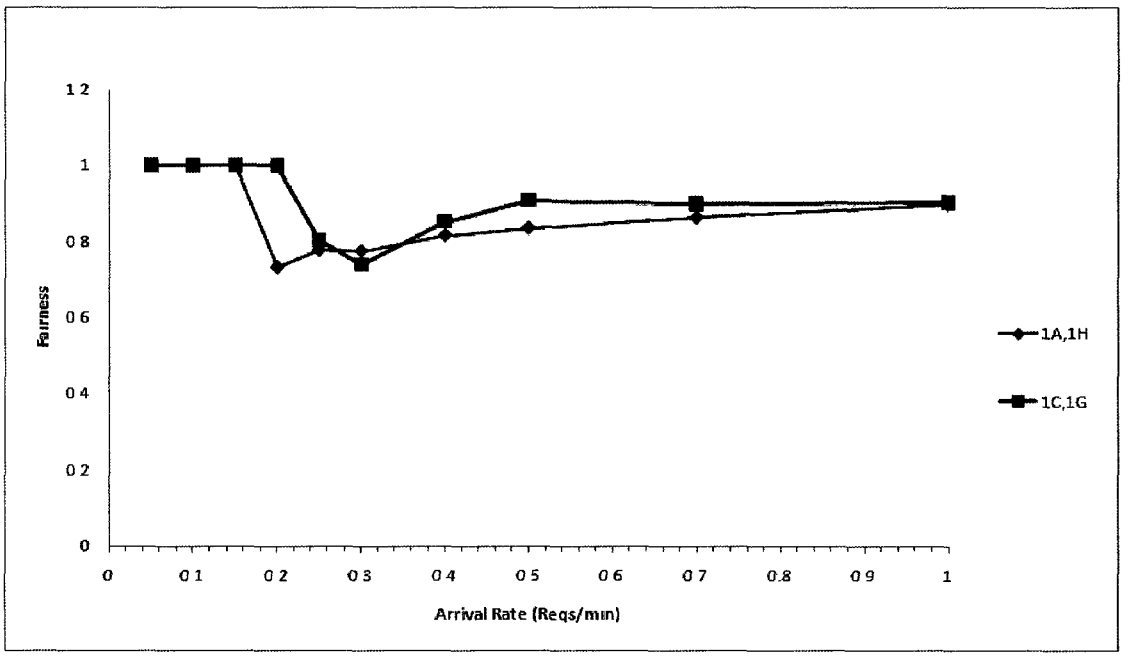

Figure 4-14: Comparison of algorithms 1A, 1H and 1C, 1G the using impact of arrival rate on Fairness (Q_R)

In figures 4-16, utilization increases as L increases. Similarly in figure 4-17, W_R is observed to be decreasing as L increases. However the performance differences in both graphs, is observed during the intermediate L values. This shows that higher performance difference can be achieved when intermediate L values are used.

In figure 4-18, it is seen that 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) outperforms 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) as Q_R values of 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) tend to be closer to 1 for most L values.

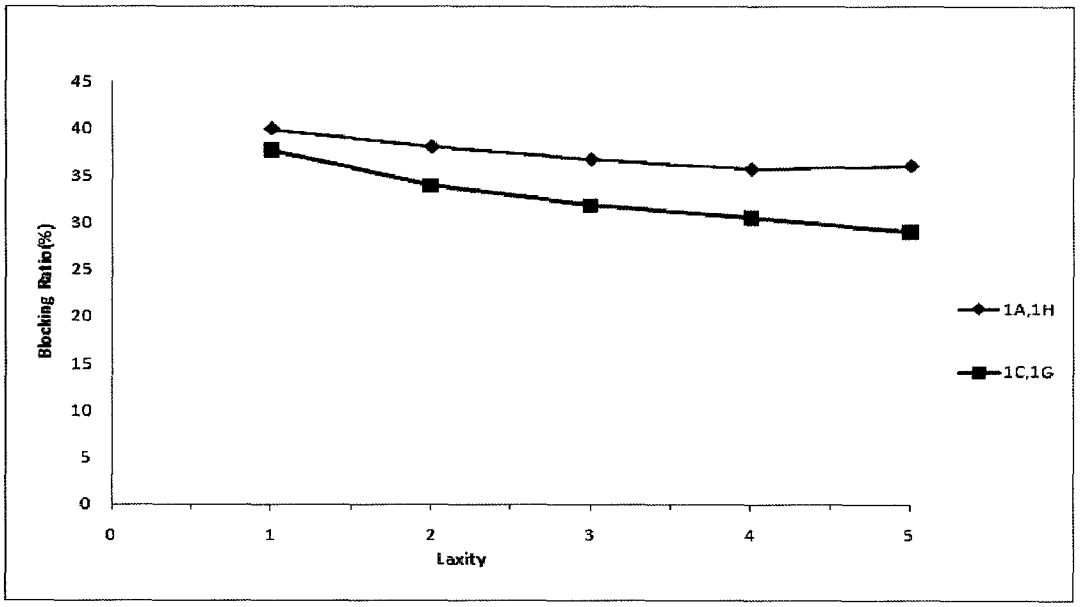

Figure 4-15: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of laxity on Blocking Ratio (B)

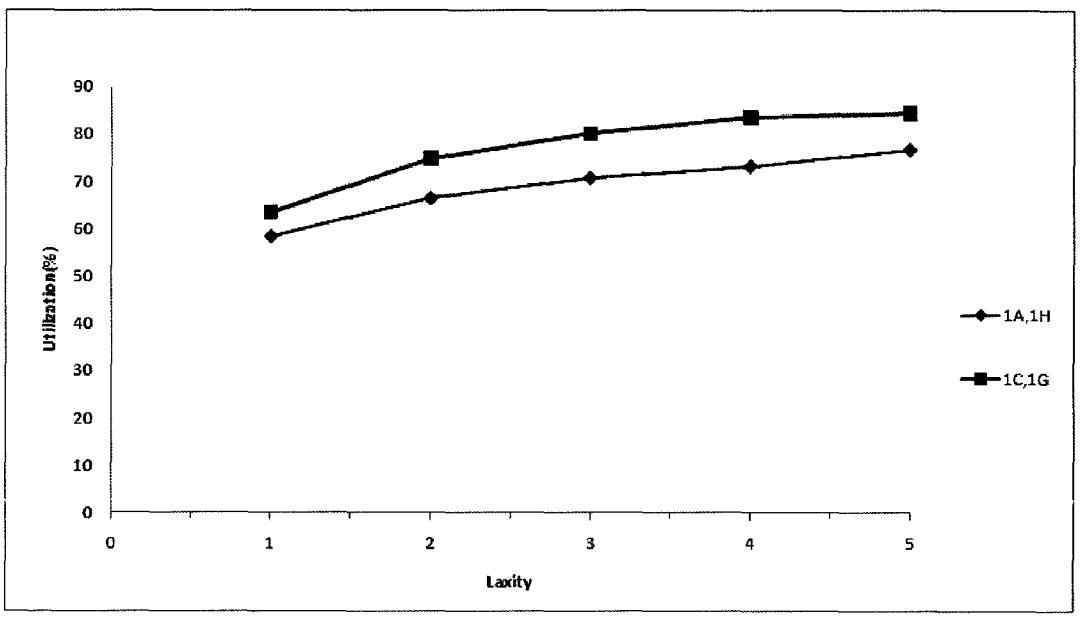

Figure 4-16: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of laxity on Utilization (U)

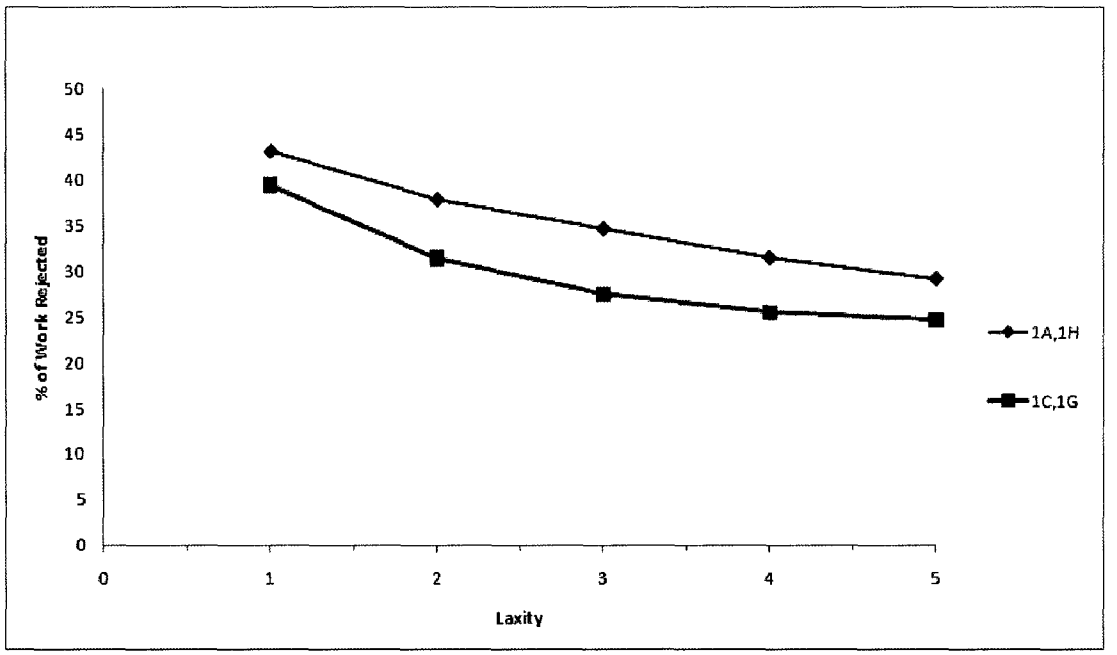

Figure 4-17: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of laxity on Percentage of Work Rejected (W_R)

4.2.3 Effect of Start Time Delay

In this section, impact of Start Time Delay on the performance of the co-allocation algorithms is analyzed. As discussed in section 3.3.3, D_{EST} is added to the arrival time of the request to get T_{EST}. As mean D_{EST} decreases, T_{EST} of the requests are squeezed together and competition between them increases because a lower mean D_{EST}, pushes the T_{EST} of the requests closer to their arrival time.

In figure 4-19, it can be observed that as competition decreases, B decreases and the gap between performance of 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) and 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) increases. This is because at higher mean D_{EST}, t_{EST} of the requests are spread in time and the flexibility to schedule them increases. When the flexibility in scheduling is higher, 1C

(First Task with Latest Available Interval), 1G (Resource with Highest Utilization) combination handles the co-allocation more effectively by choosing best-fit highly utilized resources than 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization). Similarly Figure 4-21 shows that W_R decreases as mean D_{EST} increases.

In Figures 4-21 and 4-22, it can be observed that 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) outperforms 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) for all values of mean D_{EST}.

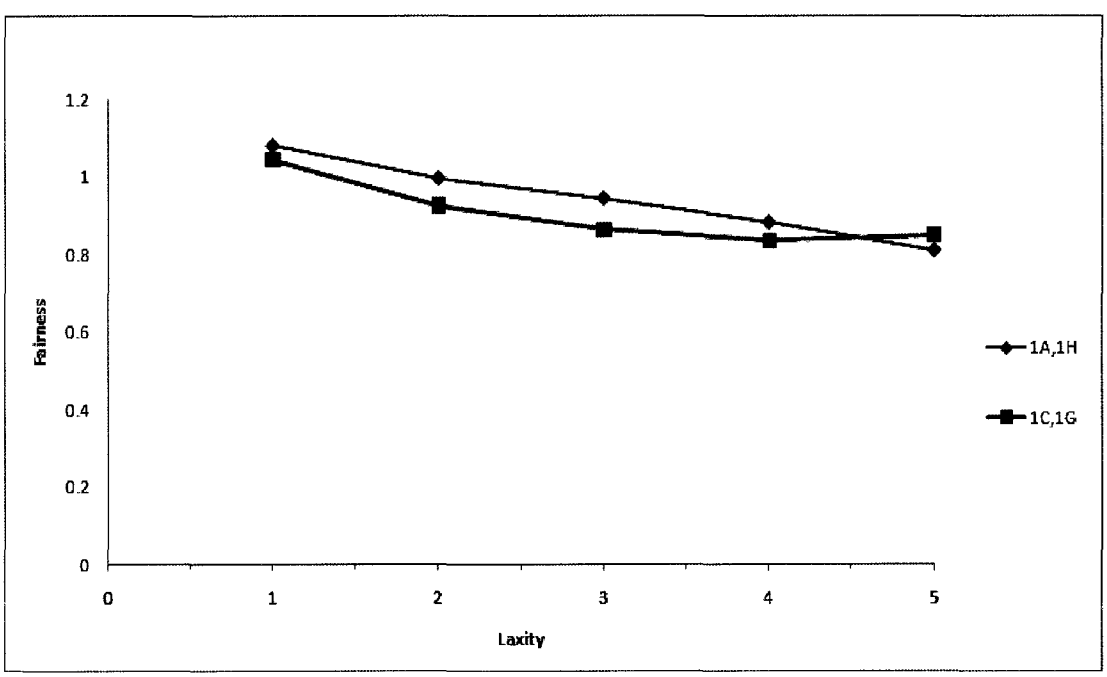

Figure 4-18: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of laxity on Fairness (Q_R)

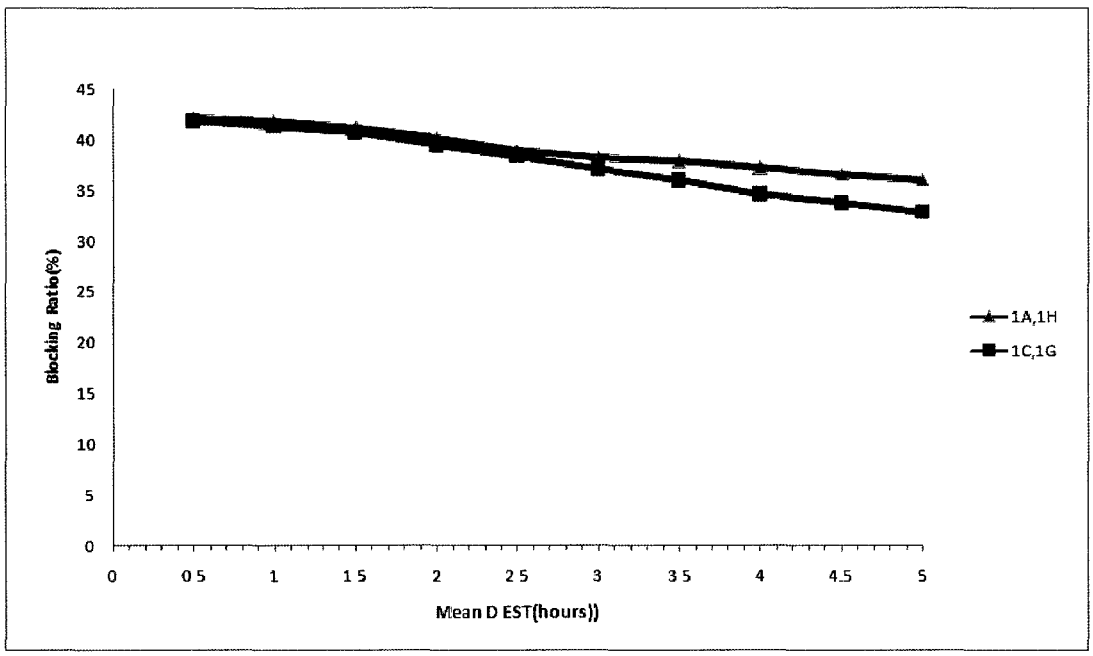

Figure 4-19: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of Start Time Delay on Blocking Ratio (B)

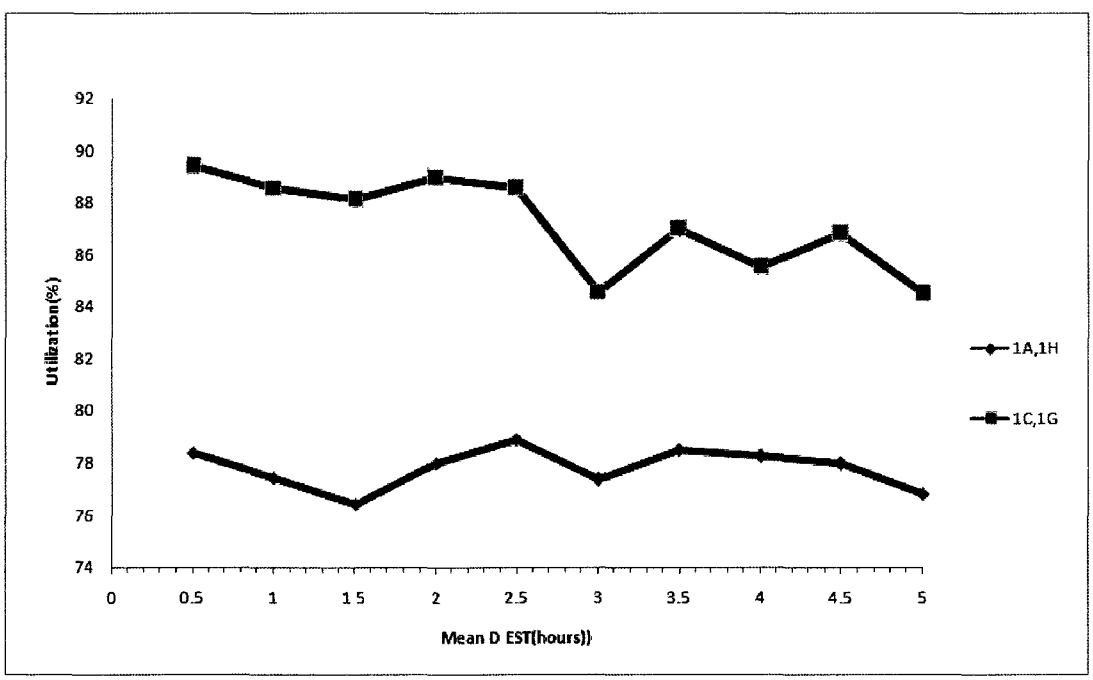

Figure 4-20: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of Start Time Delay on Utilization (U)

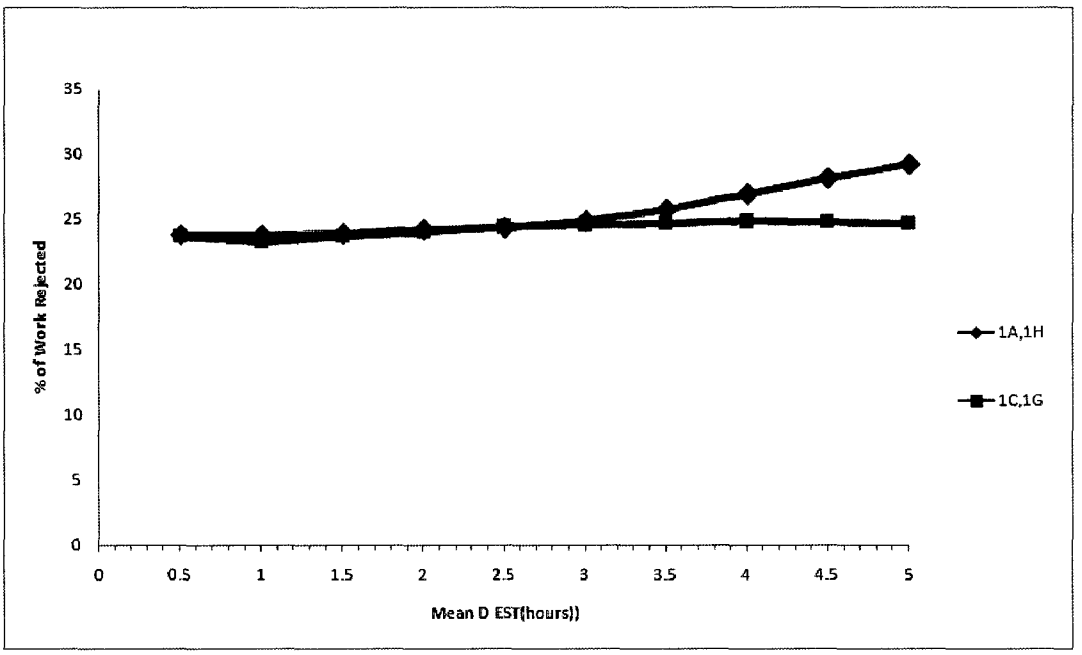

Figure 4-21: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of Start Time Delay on Percentage of Work Rejected (W_R)

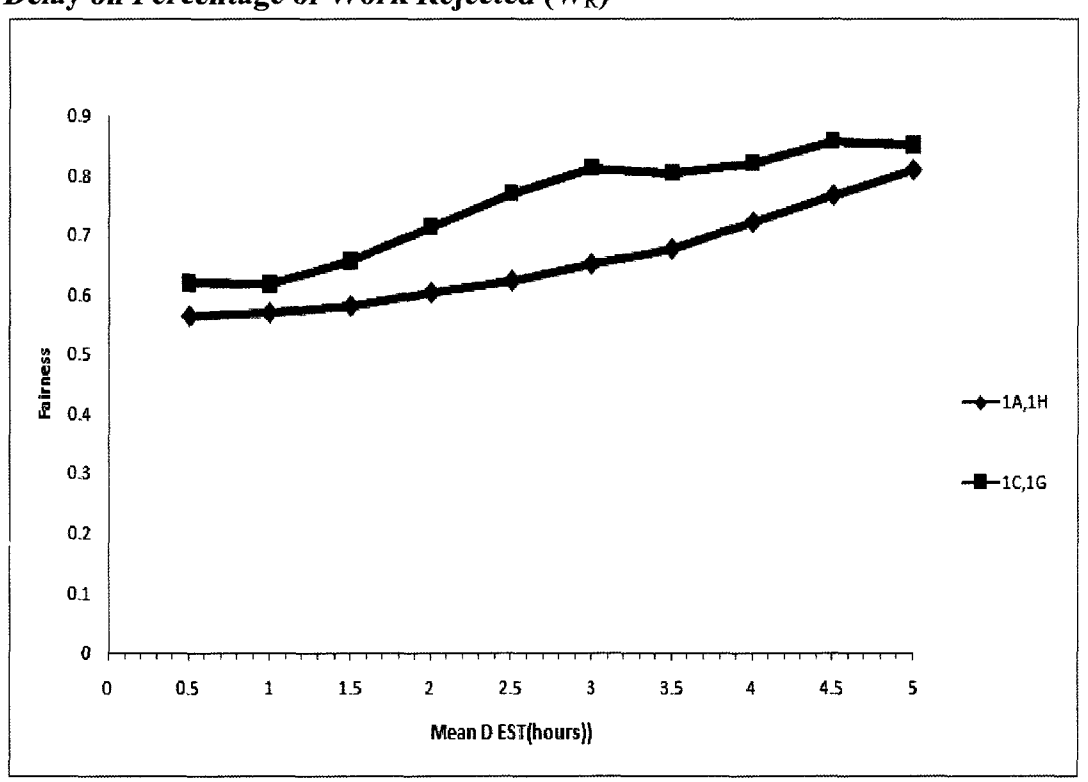

Figure 4-22: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of Start Time Delay on Fairness (Q_R)

4.2.4 Effect of the Number of Tasks

This section presents an analysis of the impact of the number of application tasks (I) on performance of the algorithms. By default, I, is uniformly distributed between 2 and 6 with a mean of 4. In this experiment, multiple uniform distributions are used to model the number of tasks and the means of these distributions are indicated on the x-axis. As seen in Figures 4-23, 4-24, 4-25 and 4-26, the variability in number of application tasks didn't have significant impact on the performance metrics.

The results show that 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) performs slightly better than 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) for all values of the number of tasks experiments for all the performance metrics.

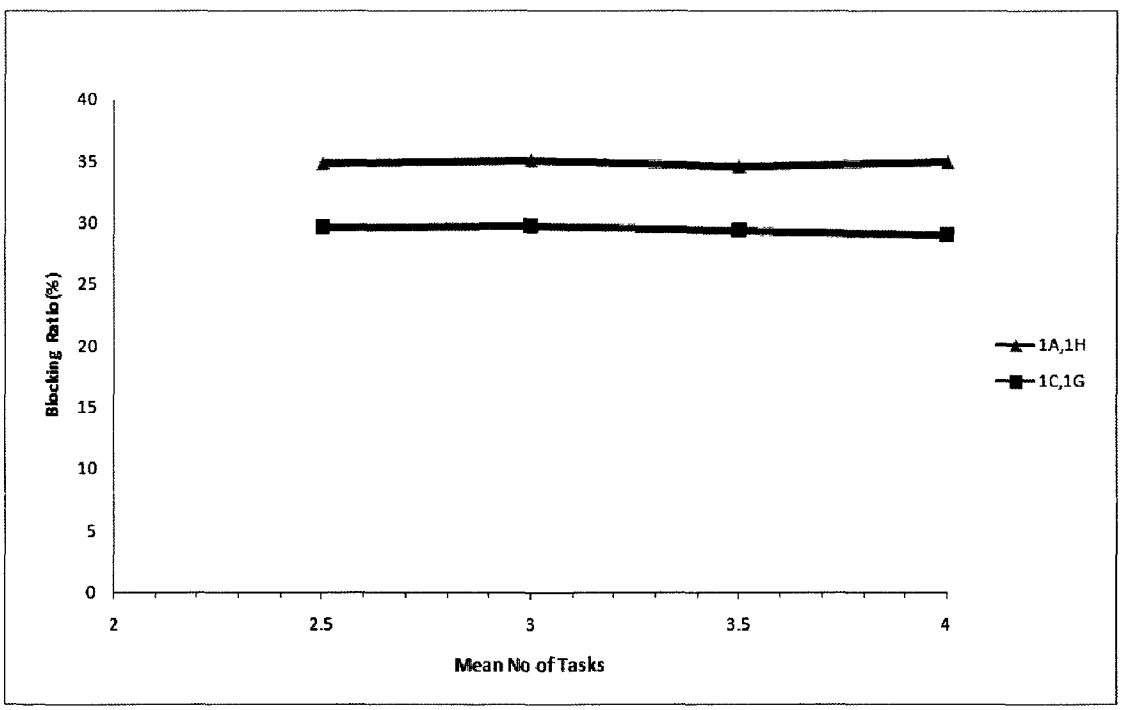

Figure 4-23: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of Number of Tasks on Blocking Ratio (B)

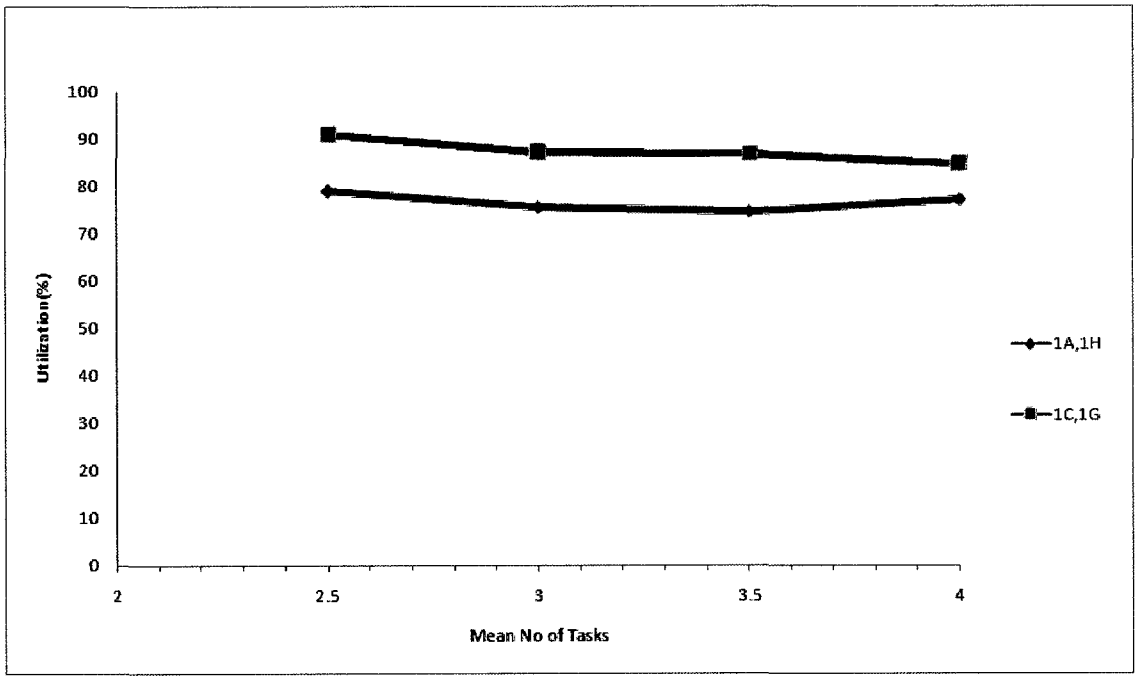

Figure 4-24: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of Number of Tasks on Utilization (U)

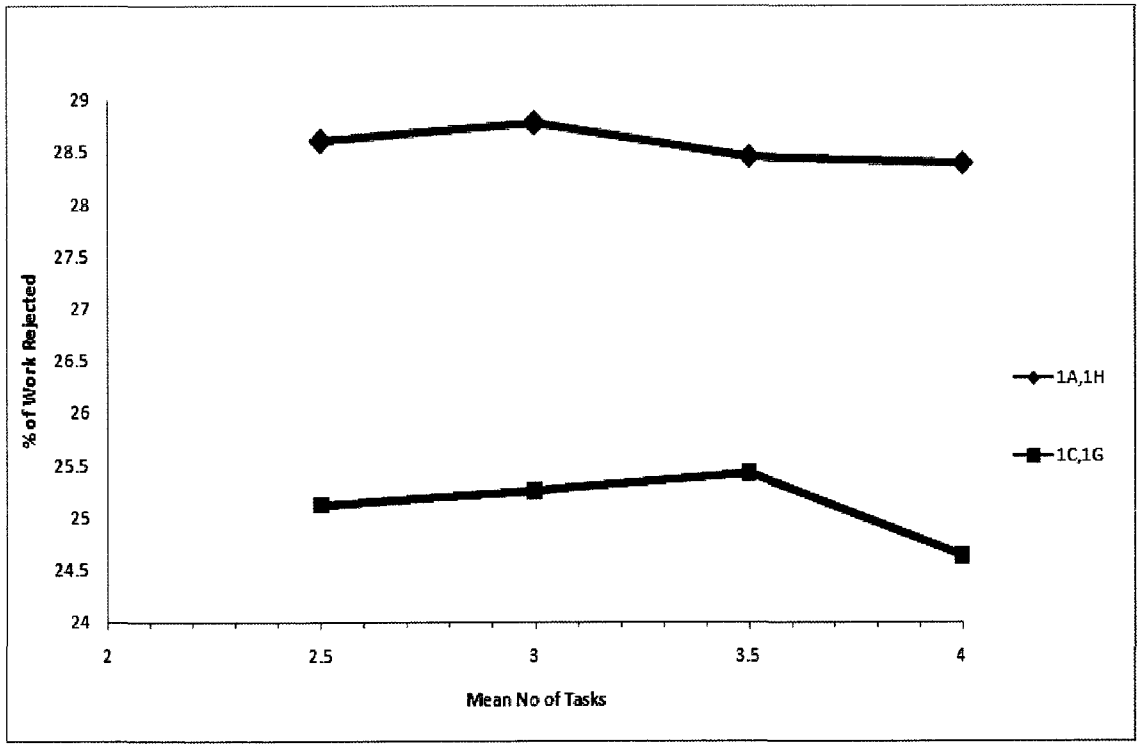

Figure 4-25: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of Number of Tasks on Percentage of Work Rejected (W_R)

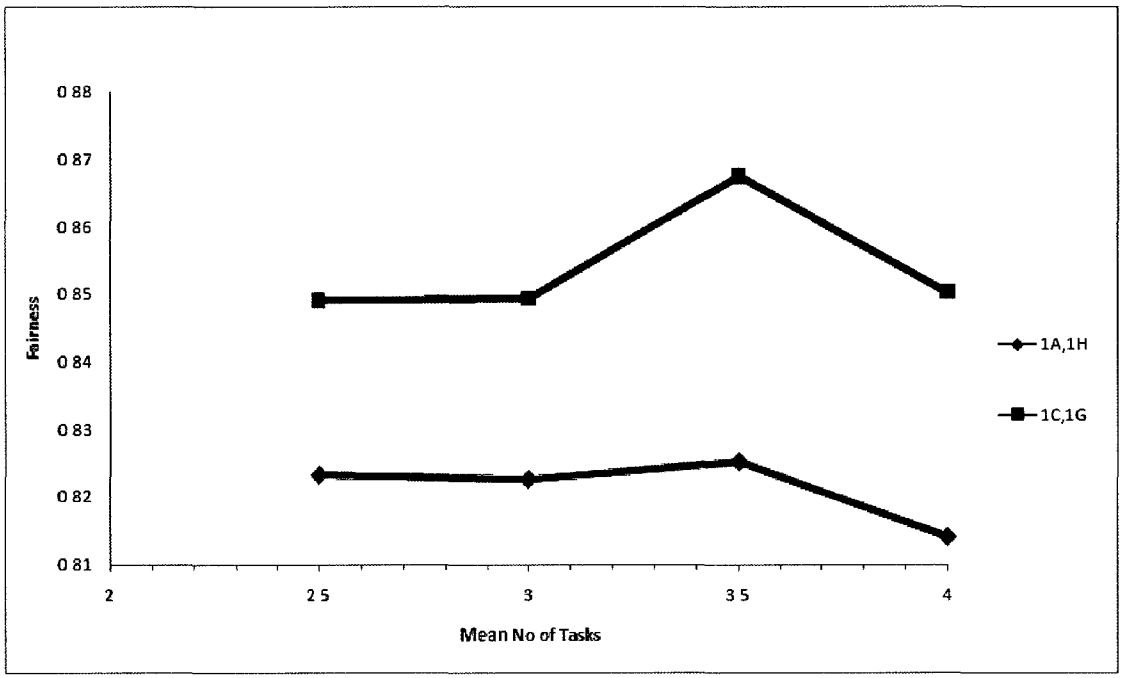

Figure 4-26: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of Number of Tasks on Fairness (Q_R)

4.2.5 Effect of Service Time

This section focuses on determining the effect of the mean application execution time (t_{ET}) on performance. In this experiment, t_{ET} is uniformly distributed and mean t_{ET} values are varied while keeping the other parameters at their default values. Uniform distribution is used to model t_{ET} values for the experiment, similar to the work in RCA-02. Mean t_{ET} is used in calculations for the distribution's upper and lower bounds. The upper and the lower bound values are obtained by multiplying 0.5 and 1.5 with mean t_{ET} respectively.

As mean t_{ET} increases, requests for larger applications are generated in the workload. Requests for larger application hold the resources for a longer period of time increasing the competition for resources. As expected, Figure 4-27 shows that as mean t_{ET} increases, B increases. Performance difference between 1C (First Task with Latest Available Interval), 1G

(Resource with Highest Utilization) and 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) is more visible for intermediate mean t_{ET} values. This is because for a lower mean service time the competition for resources is low and most requests are accepted. For very high mean service time the competition for resources is so high that a large proportion of requests is rejected and both 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) and 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) display smaller differences in system performance.

In figure 4-28, it shows that as mean t_{ET} increases, U increases. This is because longer application, gives higher utilization of resources. In figure 4-27, WR increases as the mean service time increases. However, since B increases with mean service time U seems to level off after a certain value of mean service time is reached. For intermediate mean t_{ET} values, the difference in performance between 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) and 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization), is clearly visible. Such behavior is similar to the one predicted for the system behavior captured in Figure 4.29. In Figure 4-30, fairness values of 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) tends to be closer to 1 as compared to 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization), indicating that 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) treats the large and small applications more fairly. In all the Figures 4-27 – 4-30 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) displays a higher performance as compared to 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization).

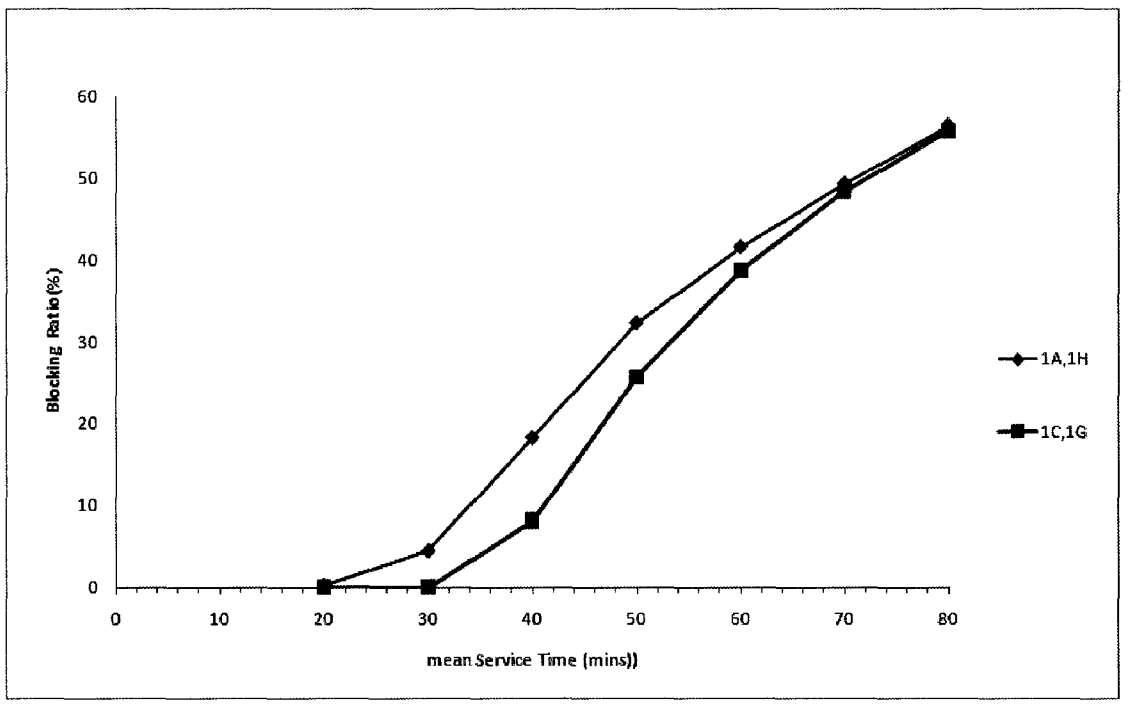

Figure 4-27: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of Mean Service Time on Blocking Ratio (B)

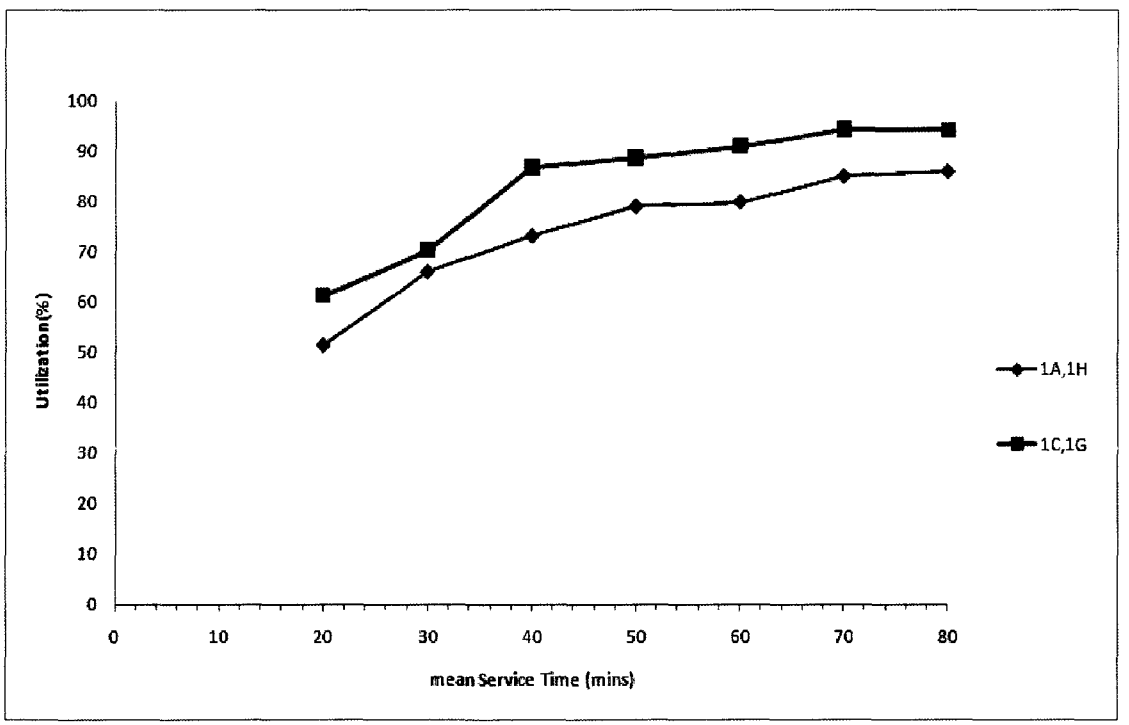

Figure 4-28: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of Mean Service Time on Utilization (U)

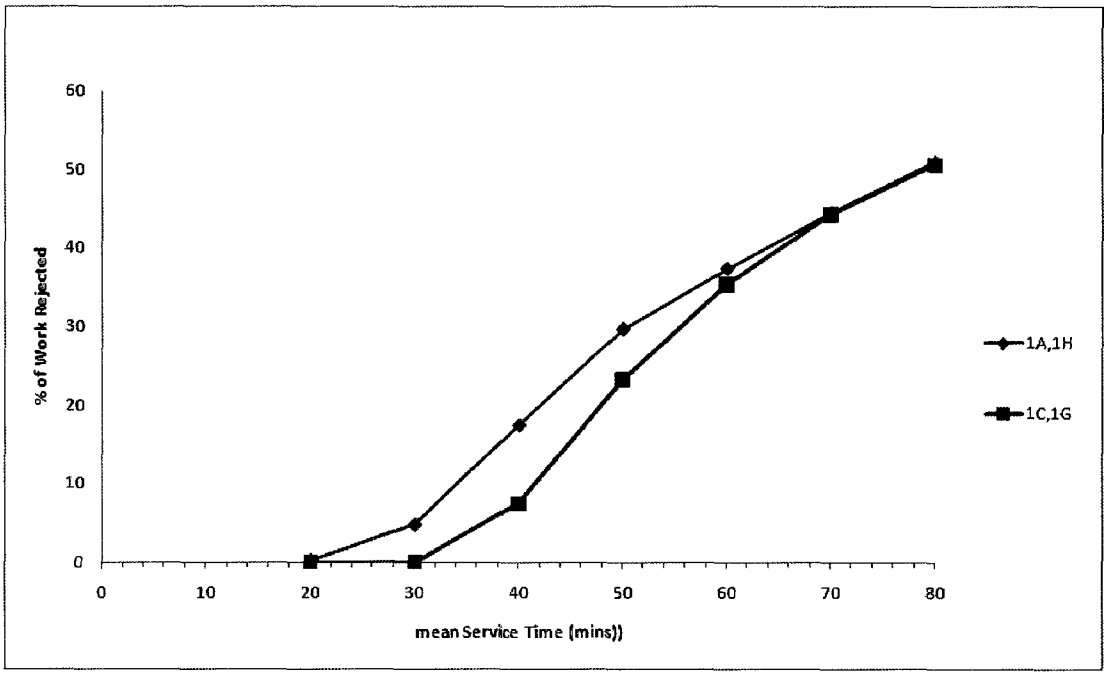

Figure 4-29: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of Mean Service Time on Percentage of Work Rejected (W_R)

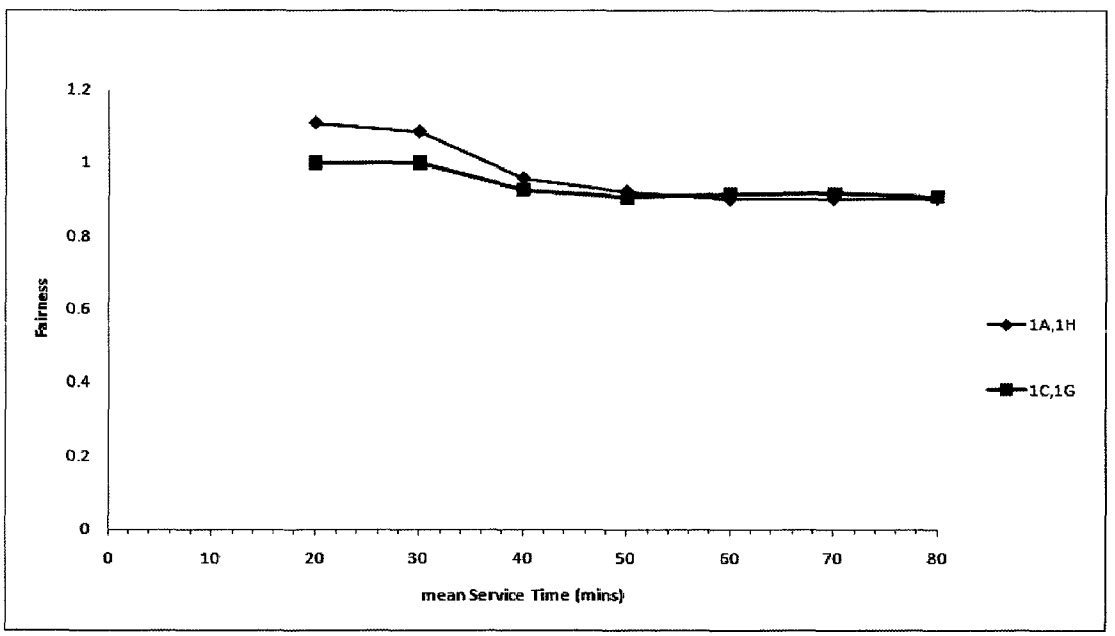

Figure 4-30: Comparison of algorithms 1A, 1H and 1C, 1G using the impact of Mean Service Time on Fairness (Q_R)

4.2.6 Performance Comparison of the Proposed Co-Allocation Algorithms with the RAN/RAN Co-Allocation Algorithm

This section discusses the performance comparison of co-allocation algorithms proposed in this thesis and the RAN/RAN algorithm presented in [FAR-07]. As discussed in section Chapter 2, the co-allocation algorithm presented in [FAR-07], uses a combination-based approach for solving the co-allocation problem. For each co-allocation request, all possible available resources combinations are derived and one combination is chosen for co-allocation. Although RAN/RAN has high computational complexity, it evaluates all possible combinations and thus has the potential of generating low B values. Thus, it was chosen as the reference for performance comparison.

Figure 4-29 shows the comparison of algorithms 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) and 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) with co-allocation algorithm, RAN/RAN. The experiment is conducted with the workload and system parameters set to their default values and the arrival rate(λ) is varied. It seems that B for RAN/RAN lies between the B of 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) and 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) for most arrival rates.

As seen in Figure 4-29, 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) outperforms RAN/RAN even though RAN/RAN examines all possible combinations for co-allocation. RAN/RAN randomly selects a feasible resource combination and does not look at the current state of the resources. However 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) uses resources state information to effectively

pick resources for co-allocation and this makes the co-allocation process to adapt the system conditions effectively. 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) provides a comparable system performance to RAN/RAN which requires higher computations in co-allocation process.

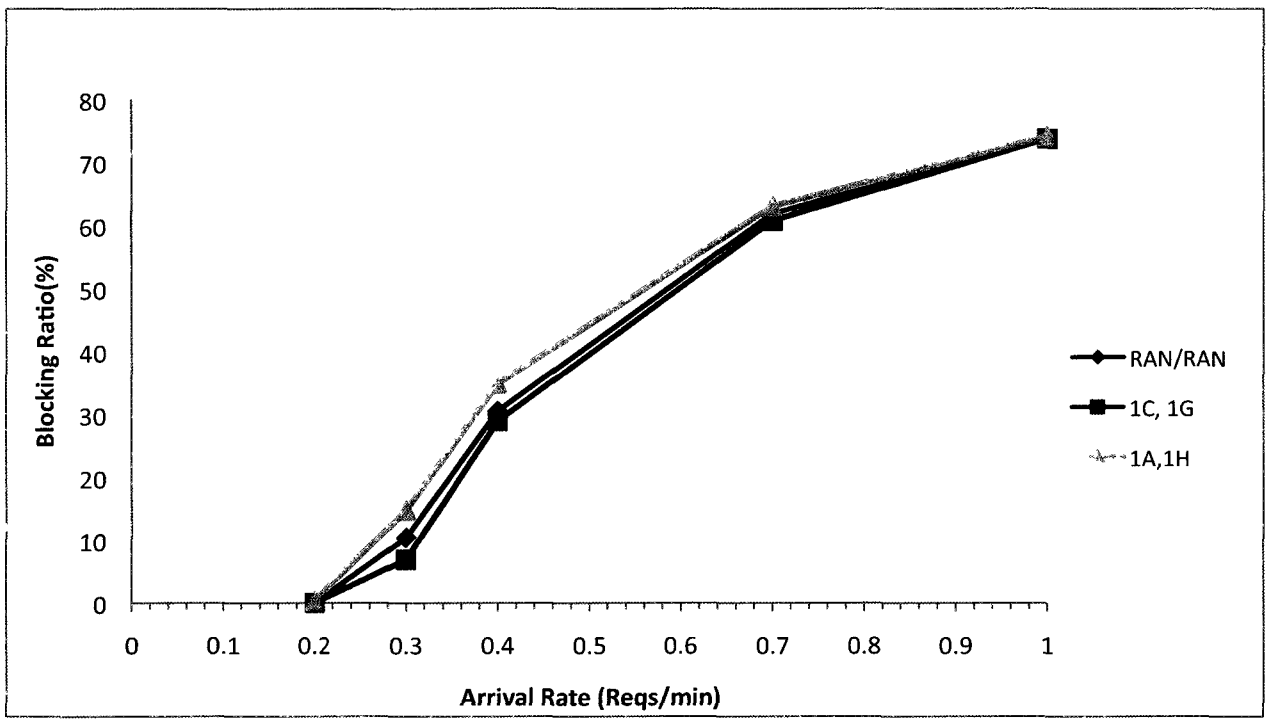

Figure 4-31: Comparison of algorithms 1A, 1H and 1C, 1G with RAN/RAN

Table 4-2: Comparison of the Algorithm Execution Time Delay of 1C,1G and 1A,1H with RAN/RAN

Algorithms	*Case 1* – Default Parameter Values (in milliseconds)	*Case 2* – High No. of Tasks (I = U(2,10)) (in milliseconds)	*Case 3* – High No. of Resources (J x M_J = 6 x 20 = 120) (in milliseconds)
RAN/RAN	183	587.8	592.467
1C,1G	122.93	207.4	227
1A,1H	119.93	209.53	225.267

Table 4-2 shows a comparison of execution times for 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) and 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) with that for RAN/RAN. Three different cases are tested. In each case, the broker entity is run on the same machine (with Processor Speed: 2.26 GHz, Memory: 4GB, OS: Windows 7). In case 1, the system and workload parameters are kept at their default values during the simulation. In case 2 and case 3 a higher Number of Task (I) and a higher Number of Resources are used respectively. All the other parameters are held at their default values. The differences in execution time between RAN/RAN and algorithms 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) and 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) seem to be high. These differences are observed to increase as the number of application tasks and number of resources increase (as seen in cases 2 and 3). The algorithmic complexity of RAN/RAN increases its execution time greatly, particularly in case 2 and 3, as compared to 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) and 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization).

4.3 Summary

The simulation experiments described in this chapter show that that the sub-algorithm combination 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) demonstrates the best performance for most combinations of the workload parameters. Although inferior, a number of other sub-algorithm combinations demonstrated a performance that is close to that of 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization). Using 1C (First Task with Latest Available Interval) in the selection of the first-task and 1G (Resource with Highest Utilization) in the selection of the next task

seems to be the most effective in making a co-allocation decision. The effect of key system and workload parameters on system performance is discussed.

Both the arrival rate and the mean task execution time t_{ET} control the system load: an increase in any of these parameters leads to a higher load on the system. The difference in B values achieved by 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) and 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) that gave rise to the best and the worst performance respectively seems to be high at intermediate values of system load. For low and high system loads all the sub-algorithm combinations are observed to perform comparably. A significant difference between the utilization, U, achieved by 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) and 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) is observed at higher system loads. A difference of approximately 7.86%, is observed for example, at λ=0.3 (see Figure 4.11. The relationship between W_R and system load follows the same pattern as B: a substantial difference between 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) and 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) is observed at intermediate system loads. For most workload parameters, 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) gives rise to the best fairness metric Q_R. By producing a Q_R close to 1, 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) treats both large and some jobs in an equitable fashion. It is interesting to note that the mean number of tasks I has minimal impact on the relative performance of 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) and 1A (First Task with Largest Interval), 1H

(Resource with Highest Utilization) (see Figure 4-23 for example). A more detailed discussion of the relationship between system and workload parameters and performance is presented in Section 5.2.

Chapter 5

Conclusions

5.1 Summary

In grids, resource co-allocation is essential in situations where applications require simultaneous availability of multiple resources throughout their execution period. Resource brokers that support co-allocation of resources need to ensure that the application tasks are scheduled to execute at the same time on different resources in potentially different domains. The broker also needs to ensure that both the resource provider's priority of optimized system performance and QoS constraints of the resource consumers are satisfied.

As discussed in Section 2.2, resource management in cloud computing share common grounds with grid computing and existing techniques for resource management on grids can be extended to cloud computing. Co-allocation algorithms and the performance analysis presented in the thesis, can also be applicable to clouds.

This thesis focuses on co-allocation allocation algorithms for Advance reservation (AR) requests with laxity. Advance reservation is used to ensure the simultaneous availability of resources and meet the QoS constraints of the applications. The thesis presents 16 different algorithms that solve the co-allocation problem effectively. Even though all the algorithms have a common flow and try to select a resource for each application task, one by one, they differ in two key steps of the co-allocation process which play crucial roles in determining the efficiency of the algorithms. One step corresponds to selecting the first-task to start the co-allocation process with. The other corresponds to selecting resources for the remaining tasks. A request is

accepted when resources for all of the application tasks are selected and they can be scheduled to execute in the same time window.

To evaluate the algorithms, a simulator is devised for comparing the performance of the various co-allocation algorithms. Workloads used in the experiments are controlled by a number of workload parameters. A factor at a time approach is used to evaluate the impact that different workload and system parameters have on the performances of the algorithms. Performance metrics used in the performance analysis, are Blocking Ratio (B), Percentage of Workload Rejected (W_R), Utilization (U) and Fairness (Q_R)

5.2 Summary of Performance Analysis

This section presents a summary of key observations made from the simulation results.

- Blocking Ratio is an important performance metric. For most workload parameters, the start-task selection algorithms, 1C (First Task with Latest Available Interval) and 1A (First Task with Largest Interval), showed the best and the worst performances in terms of B respectively. Selecting a resource with the latest available interval (1C) seems to give rise to the best system performance. Selecting a resource with the largest interval (1A) leads to the worst performance. Although inferior, performances of 1B (First Task with Smallest Interval) and 1D (First Task with Earliest Available Interval) were close to that of 1C (First Task with Latest Available Interval).
- For most workload parameters, the next-task selection sub-algorithm, 1G (Resource with Highest Utilization) and 1H (Resource with Highest Utilization), showed the best and the worst performance in terms of B respectively. Choosing a highly utilized resource (1G)

when compared to choosing a less utilized resource (1H) for an application task, gives rise to a better performance. Although inferior, the performances of 1E (Resource with Smallest Leftover Time) and 1F (Resource with Largest Leftover Time) were close to that of 1G (Resource with Highest Utilization).

- Combining both best sub-algorithms and both worst sub-algorithms respectively gives rise to the best and the worst sub-algorithm combinations, IC, 1G (Resource with Highest Utilization) and 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) respectively, as far as the Blocking Ratio is concerned.
- The performance difference between the combinations 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) and 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) is observed to widen as L increases. Allowing more flexibility in scheduling resources for application tasks lets 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) to select lower utilized resources that leads to a poorer system performance.
- It is observed that as competition decreases with an increase in start time delay, the difference between the performance of 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) and 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) increases. This is because at higher mean D_{EST}, t_{EST} of the requests are spread in time and the flexibility to schedule them increases. When the flexibility in scheduling is higher, 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) combination handles the co-allocation more effectively by choosing the latest available and highly utilized resources in

comparison to 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization).

- For intermediate mean t_{ET} values, the difference in performance between 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) and 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization), is clearly visible. When the competition for resources is low and high, acceptance of the request is influenced more by the availability of the resources than the effectiveness of the algorithms.
- Comparison of algorithms 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) and 1A (First Task with Largest Interval), 1H (Resource with Highest Utilization) with co-allocation algorithm, RAN/RAN in [FAR-07] shows that effectiveness of the proposed algorithms are comparable to a combination based co-allocation algorithm in which all available resource combinations are determined before selecting resources for the application tasks. 1C (First Task with Latest Available Interval), 1G (Resource with Highest Utilization) shows a slightly better performance than RAN/RAN.

5.3 Future Research

Directions for future research include the following:

- The proposed co-allocation algorithms do not consider network delays when choosing the resources for co-allocation. Incorporating network delays in resource selection can potentially lead to a more effective co-allocation.

- The co-allocation process considered in this thesis assumes that the user estimated execution time is correct and makes decision accordingly. This may not be always true in real systems. Handling uncertainties associated with the user estimated application execution time is an important direction for future research.
- The co-allocation algorithms presented in this thesis, service AR co-allocation requests. Supporting on demand co-allocation in addition to ARs warrants further investigation.

References

[AMA-1] Amazon Simple Storage Service, "Amazon S3", http://aws.amazon.com/s3/, Accessed December 2010.

[AMA-2] Amazon Elastic Block Store (EBS) , "Amazon EBS", http://aws.amazon.com/ebs/, Accessed December 2010.

[AMA-3] Amazon Elastic Compute Cloud, "Amazon EC2", http://aws.amazon.com/ec2/, Accessed December 2010.

[ARM-09] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing", Technical Report No. UCB/EECS-2009-28, Dept. of Electrical and Computer Eng., University of California, Berkeley, CA, USA, February 2009.

[BAL-03] Z. Balaton and G. Gombás, "Resource and Job Monitoring in the Grid", In Proceedings of 9th International Euro-Par Conference, pp. 404-411, Klagenfurt, Austria, August, 2003.

[BUY-00] R. Buyya, D. Abramson, and J. Giddy, "An Economy Driven Resource Management Architecture for Global Computational Power Grids", In Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), pp.26-29, Las Vegas, NV, USA, June, 2000.

[BUY-02] R. Buyya, D. Abramson , J. Giddy , H. Stockinger, "Economic Models for Resource Management and Scheduling in Grid Computing", International Journal of Concurrency and Computation: Practice and Experience, Vol.14, pp. 1507-1742, 2002.

[BUY-09] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility", in Future Generation Computer Systems, 25(6): 599-616, June, 2009.

[BUY1-02] R. Buyya, M. Murshed, "GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing", in Concurrency and Computation: Practice and Experience, 14(13-15): 1175-1220, December, 2002.

[BUY2-00] R. Buyya, D. Abramson, "An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications", In proceedings of the 2nd International Workshop on Active Middleware Services(AMS 2000), pp.221-230, Pittsburgh, PA, USA, August, 2000

[BUY3-00] R. Buyya, D. Abramson, J. Giddy, "Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid," In Proceedings of the 4th International Conference and Exhibition on High Performance Computing in Asia-Pacific Region, pp. 283-289, Beijing, China, May 2000.

[CAS-00] H. Casanova, F. Berman, G. Obertelli, R. Wolski, "The AppLes Parameter Sweep Template: User-Level Middleware for the Grid", In Proceedings of the 2000 ACM/IEEE conference on Supercomputing, pp.60, Dallas, TX, USA, November, 2000.

[CAS-09] C. Castillo, G. N. Rouskas, K. Harfoush,"Resource Co-Allocation for Large-Scale Distributed Environments", In Proceedings of the 18th ACM international symposium on High performance distributed computing, (HPDC'09), pp. 131-140, Munich, Germany, June, 2009.

[CZA-98]K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, "A Resource Management Architecture for Metacomputing Systems," In Proceedings of the 4th IPPS/SPDP Workshop on Job Scheduling Strategies for Parallel Processing, pp. 62-82, Orlando, FL, USA, March 1998.

[DEC-07] J. Decker, J. Schneider, "Heuristic Scheduling of Grid Workflows Supporting Co-Allocation and Advance Reservation", In Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid, (CCGrid2007), pp. 335-342, Rio de Janeiro, Brazil, May, 2007.

[FAR-07] U. Farooq, "A Framework for Quality of Service Aware Resource Management in Multi-Institutional Grids," Ph.D. dissertation, Dept. of Systems and Computer Eng., Carleton University, Ontario, Canada, 2007.

[FOS-1] I. Foster and C. Kesselman, "Computational Grids", 1998, http://www.globus.org/alliance/publications/papers/chapter2.pdf, Accessed November 2010.

[FOS-02] I. Foster, "What is the Grid? A Three Point Checklist," in GRID Today, July 2002. http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf. Accessed October 2010.

[FOS-03] I. Foster , J. Vöckler , M. Wilde , Y. Zhao, "The virtual grids: a New Model and Architecture for Data intensive collaboration", In Proceedings of the 15th International Conference on Scientific and Statistical Database Management. (SSDBM 2003), pp.11-, Cambridge, MA, USA, July 2003.

[FOS2-99] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, A. Roy, "A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation", In Proceedings of the International Workshop on Quality of Service, pp. 27-36, London, UK, May, 1999.

[GMDS-1] Globus Alliance, GT Information Services: Monitoring & Discovery System. http://www.globus.org/toolkit/mds/, Accessed October 2010.

[GSAM-1] GridSAM Project, "GridSAM architecture guide", http://gridsam.sourceforge.net/2.0.1/architectureguide.html, Accessed October 2010.

[GUE-00] R. Guerin, A. Orda, "Networks with Advance Reservations: The Routing Perspective". In Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies, (INFOCOM 2000), pp. 118-127, Tel-Aviv, Israel, March, 2000.

[IAM-01] A. Iamnitchi and I. Foster, "On Fully Decentralized Resource Discovery in Grid Environments," in the Proceeding of the IEEE International Workshop on Grid Computing, pp.51-62, Denver, CO,USA, November, 2001.

[JOS-1] J. Joseph and C. Fellenstein, Grid Computing, Prentice Hall PTR, 2004, pp.55-78

[KRA-1] K. Krauter, R. Buyya, M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing", http://www.buyya.com/papers/gridtaxonomy.pdf, Accessed October 2010.

[LAM-1] L. Lamport, "Paxos Made Simple", November, 2001, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.69.3093&rep=rep1&type=pdf, Accessed November 2010.

[LAV-04] T. Lavian, J. Mambretti, D. Cutrell, H. Cohen, S. Merrill, R. Durairaj, P. Daspit, I. Monga, S. Naiksatam, S. Figueira, D. Gutierrez, D. Hoang, F. Travostino, "DWDM-RAM: a data intensive Grid service architecture enabled by dynamic optical networks", In Proceedings of the 4th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2004), pp.762-764, Chicago, IL, USA, April, 2004.

[LIT-61] J. D. C. Little, "A Proof for The Queuing Formula", Operations Research, Vol. 9, No. 3, pp. 383-387, 1961.

[MAC-11] J. MacLaren, M. Mc Keown, "HARC: A Highly-Available Robust Co-scheduler", www.realitygrid.org/publications/HARC.pdf, Accessed January 2011.

[MAJ-09] 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 (International Workshop on Cloud Computing), pp.490-495, Shanghai, China, May, 2009.

[MAT-1] L. Matyska, A. K˘renek, M. Ruda, M. Vocu, Z. Salvet, J. Sitera, J. Pospíšil, D. Kouril, "The grid job monitoring service", http://tnc2002.terena.org/Papers/p7b4-matyska.pdf, Accessed November 2010.

[PLA-02] B. Plale, P. Dinda, and G. von Laszewski, "Key Concepts and Services of a Grid Information Service", In Proceedings of the 15th International Conference on Parallel and Distributed Computing Systems (PDCS 2002), pp.437-442, Cambridge, MA, USA, November, 2002.

[RAM-98] R. Raman, M. Livny, M.Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing", In Proceedings of the 7th International Symposium on High Performance Distributed Computing, pp.140-146, Chicago, IL , USA, July, 1998.

[REI-95] W. Reinhardt, "Advance Resource Reservation and its impact on Reservation Protocols", In Proceedings of Broadband Islands '95, pp.28-35, Dublin, Ireland, September, 1995.

[SCH-98] A. Schill, F. Breiter, S. Kuhn, "Design and Evaluation of an Advance Reservation Protocol on Top of RSVP", In Proceedings of the 4th International Conference on Broadband Communications (BC'98), pp.23-40, Stuttgart, Germany, 1998.

[SID-06] M. Siddiqui, A. Villazon, T. Fahringer", Grid Capacity Planning with Negotiation-based Advance Reservation for Optimized QoS", In Proceedings of the 2006 ACM/IEEE conference on Supercomputing, pp.21, Tampa, Florida, November, 2006.

[SNM-07] OO Sonmez , A. Gursoy, "A Novel Economic-Based Scheduling Heuristic for Computational Grids", Int. Journal of High Performance Computing Applications, vol. 21, no. 1, pp.21-29, February 2007.

[TAK-07] A. Takefusa, H. Nakada, T. Kudoh, Y. Tanaka, "GridARS: An Advance Reservation-based Grid Co-allocation Framework for Distributed Computing and Network Resources", In Proceedings of the 13th International Workshop on Job Scheduling Strategies for Parallel Processing, pp. 152-168, Seattle, WA, USA , June, 2007.

[TER-1] TeraGrid, "TeraGrid", https://www.teragrid.org/web/about/ Accessed January 2011.

[TOP-02] H. Topcuoglu, S. Hariri, M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," IEEE Transactions on Parallel and Distributed Systems, pp. 260-274, vol. 13, no. 3, 2002.

[UKNG-1] The National Grid Service (NGS), "UK National Grid Service", http://www.ngs.ac.uk/, Accessed November 2010.

[WAN-03] L. Wang, W. Cai, B. Lee, W. Jie, "Resource Co-allocation for Parallel Tasks in Computational Grids", In Proceedings of the International Workshop on Challenges of Large Applications in Distributed Environments (CLADE2003), pp.88, Seattle, WA, USA, June, 2003

[WIL-1] R. J. Wilson, "The European DataGrid Project", 2001, http://www.snowmass2001.org/e7/papers/wilson.pdf, Accessed November 2010.

[XIN-04] J. Xing, C. Wu, M. Tao, L. Wu, H. Zhang,"Flexible Advance Reservation for Grid Computing", In Proceedings of Grid and Cooperative Computing (GCC'2004). pp.241-248, Wuhan, China, October 2004

[ZHA-06] W. Zhang, A.M.K. Cheng, M. Hu, "Multisite Co-allocation algorithms for computational grid", In Proceedings of Parallel and Distributed Processing Symposium (IPDPS'06), pp.395, Rhodes Island, Greece, April, 2006.

Appendix

Table A.1: Comparison of Blocking Ratios for Various Arrival Rates (1A, 1E to 1B, 1H)

Arrival Rate (λ)	1A, 1E	1A, 1F	1A, 1G	1A, 1H	1B, 1E	1B, 1F	1B, 1G	1B, 1H
0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.15	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.2	0.24	0.22	0.30	0.34	0.00	0.04	0.00	0.00
0.25	4.16	4.14	4.44	4.48	0.84	0.86	0.72	0.98
0.3	14.66	14.14	14.08	14.78	7.62	7.92	6.90	8.70
0.4	34.48	34.44	34.36	34.88	32.42	32.14	29.06	33.06
0.5	47.52	47.56	47.30	48.00	46.58	46.56	43.66	46.74
0.7	62.78	62.56	62.42	63.08	62.32	62.46	60.84	62.96
1	73.88	73.82	73.68	74.16	73.92	73.94	73.24	74.16

Table A.2: Comparison of Blocking Ratios for Various Arrival Rates (1C, 1E to 1D, 1H)

Arrival Rate (λ)	1C, 1E	1C, 1F	1C, 1G	1C, 1H	1D, 1E	1D, 1F	1D, 1G	1D, 1H
0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.2	0.00	0.00	0.00	0.02	0.04	0.06	0.00	0.00
0.25	0.66	0.76	0.76	1.32	1.18	1.48	0.96	1.40
0.3	7.68	7.94	6.92	8.96	10.40	9.72	9.66	10.20
0.4	32.16	32.42	28.96	32.78	33.16	33.20	32.18	33.54
0.5	46.26	46.60	42.68	46.94	46.82	47.14	46.34	47.26
0.7	62.40	62.32	60.80	62.70	62.70	62.68	62.74	62.88
1	73.86	73.72	73.80	74.20	74.04	74.00	73.92	74.38

Table A.3: Comparison of Blocking Ratios for Various Laxity Values (1A, 1E to 1B, 1H)

Laxity	1A, 1E	1A, 1F	1A, 1G	1A, 1H	1B, 1E	1B, 1F	1B, 1G	1B, 1H
1	39.12	38.9	39.92	39.93	38.48	38.6	38.45	38.52
2	37.44	36.98	37.76	37.98	34.8	35.15	34.7	34.97
3	35.46	35.5	36.53	36.68	33.22	33.17	32.75	33.23
4	34.4	33.98	35.63	35.65	32.95	33	31.62	33.67
5	34.48	34.44	35.53	36	33.66	33.37	30.32	33.06

Table A.4: Comparison of Blocking Ratios for Various Laxity Values (1C, 1E to 1D, 1H)

Laxity	1C, 1E	1C, 1F	1C, 1G	1C, 1H	1D, 1E	1D, 1F	1D, 1G	1D, 1H
1	37.6	37.7	37.68	37.66	37.6	37.64	37.24	37.14
2	33.82	33.72	33.86	34.04	35	34.6	34.72	34.76
3	32.06	31.9	31.78	31.98	33.1	33.18	33.04	33.38
4	31.62	31.9	30.46	32.48	32.88	32.46	31.98	33.26
5	32.16	32.42	28.96	32.78	33.16	33.2	32.18	33.54

Table A.5: Comparison of Blocking Ratios for Various Start Time Delay Values (1A, 1E to 1B, 1H)

Start Time Delay	1A, 1E	1A, 1F	1A, 1G	1A, 1H	1B, 1E	1B, 1F	1B, 1G	1B, 1H
1	40.23	41.21	41.10	42.12	40.52	40.52	38.64	41.76
2	39.40	40.87	40.58	41.82	40.18	40.18	36.70	41.32
3	38.40	40.18	39.60	41.10	39.34	39.50	35.66	40.60
4	37.93	39.55	38.80	40.15	38.48	38.54	34.12	39.76
5	37.23	38.68	37.97	38.93	37.40	37.50	31.74	38.42
6	37.03	37.55	37.30	38.18	36.18	36.42	31.48	36.78
7	36.30	37.12	36.88	37.87	34.88	34.90	30.96	35.82
8	35.68	36.83	36.67	37.17	33.80	33.80	30.18	34.52
9	35.10	36.08	35.97	36.60	32.84	32.98	29.34	33.74
10	34.00	35.60	35.53	36.00	32.42	32.14	29.06	33.06

Table A.6: Comparison of Blocking Ratios for Various Start Time Delay Values (1C, 1E to 1D, 1H)

Start Time Delay	1C, 1E	1C, 1F	1C, 1G	1C, 1H	1D, 1E	1D, 1F	1D, 1G	1D, 1H
1	40.48	40.36	38.18	41.78	40.86	40.76	40.38	41.70
2	40.20	40.20	37.66	41.32	40.32	40.26	39.74	41.30
3	39.42	39.42	36.06	40.66	39.58	39.66	38.76	40.44
4	38.52	38.58	33.70	39.40	38.62	38.68	38.06	39.40
5	37.28	37.40	31.68	38.38	37.40	37.64	36.90	38.32
6	36.18	36.12	30.24	37.02	36.34	36.40	35.84	37.18
7	34.94	34.84	30.64	35.92	35.82	35.50	35.14	35.80
8	34.02	33.84	30.24	34.52	35.00	34.52	33.54	35.10
9	33.06	32.92	28.90	33.68	33.90	33.64	33.12	34.08
10	32.16	32.42	28.96	32.78	33.16	33.20	32.18	33.54

Table A.7: Comparison of Blocking Ratios for Various Number of Task Values (1A, 1E to 1B, 1H)

No. of Tasks	1A, 1E	1A, 1F	1A, 1G	1A, 1H	1B, 1E	1B, 1F	1B, 1G	1B, 1H
U(2,3)	34.90	34.70	34.18	34.74	33.70	33.30	29.58	32.90
U(2,4)	34.30	34.44	34.14	34.98	33.58	33.80	29.72	32.80
U(2,5)	34.68	34.34	34.22	34.48	33.50	33.53	29.3	32.92
U(2,6)	34.48	34.44	34.36	34.88	33.62	33.37	28.96	33.06

Table A.8: Comparison of Blocking Ratios for Various Number of Task Values (1C, 1E to 1D, 1H)

No. of Tasks	1C, 1E	1C, 1F	1C, 1G	1C, 1H	1D, 1E	1D, 1F	1D, 1G	1D, 1H
U(2,3)	32.34	32.42	29.30	32.66	34	33.78	33.54	34.48
U(2,4)	32.42	32.58	29.22	32.84	33.66	33.42	33.08	33.92
U(2,5)	32.22	32.34	28.80	32.58	33.32	33.32	32.9	33.66
U(2,6)	32.16	32.42	29.06	32.78	33.16	33.2	32.18	33.54

Table A.9: Comparison of Blocking Ratios for Various Service Time Values (1A, 1E to 1B, 1H)

Mean Service Time	1A, 1E	1A, 1F	1A, 1G	1A, 1H	1B, 1E	1B, 1F	1B, 1G	1B, 1H
20	0.22	0.18	0.24	0.20	0.00	0.00	0.00	0.00
30	4.48	4.94	4.46	4.40	0.02	0.02	0.00	0.02
40	18.36	18.40	18.22	18.24	7.86	8.30	8.08	8.88
50	32.04	32.24	32.12	32.20	26.72	26.60	25.66	27.06
60	41.32	41.20	41.16	41.50	39.42	39.42	38.66	39.60
70	48.90	48.92	48.88	49.28	48.92	48.94	48.32	49.20
80	56.18	56.14	56.08	56.36	56.30	56.28	55.74	56.22

Table A.10: Comparison of Blocking Ratios for Various Service Time Values (1C, 1E to 1D, 1H)

Mean Service Time	1C, 1E	1C, 1F	1C, 1G	1C, 1H	1D, 1E	1D, 1F	1D, 1G	1D, 1H
20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	0.00	0.00	0.02	0.00	0.16	0.26	0.20	0.38
40	8.04	8.14	8.00	8.74	11.34	11.34	10.92	11.18
50	26.56	26.50	25.58	26.96	28.46	28.20	28.10	28.36
60	39.38	39.30	38.62	39.54	40.44	40.00	39.84	40.28
70	48.92	49.02	48.10	48.96	48.98	49.00	49.00	49.22
80	56.26	56.26	55.74	56.34	56.30	56.42	56.20	56.46