SWIFT GAP: A NEW SCHEDULING MECHANISM WITH COMPLETION TIME SCHEME USING BACKFILLING TECHNIQUE FOR GRID COMPUTING ENVIRONMENT

OMAR DAKKAK

DOCTOR OF PHILOSOPHY
UNIVERSITI UTARA MALAYSIA
2017

Permission to Use

In presenting this thesis in fulfilment of the requirements for a postgraduate degree from Universiti Utara Malaysia, I agree that the Universiti Library may make it freely available for inspection. I further agree that permission for the copying of this thesis in any manner, in whole or in part, for scholarly purpose may be granted by my supervisor(s) or, in their absence, by the Dean of Awang Had Salleh Graduate School of Arts and Sciences. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to Universiti Utara Malaysia for any scholarly use which may be made of any material from my thesis.

Requests for permission to copy or to make other use of materials in this thesis, in whole or in part, should be addressed to:

Dean of Awang Had Salleh Graduate School of Arts and Sciences
UUM College of Arts and Sciences
Universiti Utara Malaysia
06010 UUM Sintok

Abstrak

Pengkomputeran grid ialah infrastruktur pengkomputeran bersambung yang memberikan akses yang dipercayai, stabil, sentiasa ada dan ekonomi untuk kuasa pengkomputeran atasan. Sifat dinamik grid membawa beberapa cabaran kepada konsep yang sedia ada dalam algoritma penjadualan berasaskan giliran.Pendekatan ini biasanya melakukan penjadualan berdasarkan keutamaan tetap tertentu yang menyebabkan peningkatan kelewatan untuk aplikasi berjalan.Oleh itu, prestasi keseluruhan akan merosot dengan ketara. Tujuan utama kajian ini adalah untuk meminimumkan kelewatan dalam penjadual bagi tugas dinamik yang. Oleh itu, penyelidikan ini menangani isu penjadualan dinamik dengan mencadangkan mekanisme Swift Gap (SG). SG terdiri daripada dua peringkat dengan menggunakan dua mekanisme: Best Gap (BG) dan Tabu Search (TS). Pada peringkat pertama, tugas diletakkan di jurang terawal yang terdapat dalam jadual sumber daya tempatan, sementara tahap kedua mengoptimumkan prestasi dengan memeriksa semua jurang yang ada di kalangan jadual sumber untuk mendapatkan jurang yang lebih baik untuk diletakkan tugas. Untuk terus meningkatkan prestasi, Skim Masa Penyiapan (CTS) dibangunkan. CTS mengurangkan kelewatan meletakkan tugas dalam jurang yang menjamin masa permulaan yang terbaik untuk tugas, dan sumber terpantas yang ada.Integrasi antara SG dan CTS (SG-CTS) dicapai dengan menggunakan aturan permulaan terbaik pada peringkat pertama sahaja, sedangkan tahap kedua termasuk kedua-dua peraturan. SG-CTS dinilai secara numerik melalui simulasi dengan menggunakan beban kerja sebenar yang mencerminkan persekitaran sistem grid nyata. Hasil kajian menunjukkan bahawa SG-CTS meningkatkan keperlahanan sebanyak 27%, keperlahanan terbatas sebanyak 25%, kelewatan sebanyak 21%, masa menunggu sebanyak 16% dan masa tindak balas sebanyak 7% berbanding dengan mekanisme Isian belakang yang konservatif diikuti oleh Carian Jarak (CONS- GS). Sumbangan kajian ini dijangka dapat membantu pengguna akhir dengan ketara dengan menyediakan penjadualan yang lebih baik dalam persekitaran pengkomputeran grid, terutamanya untuk aplikasi Pengkomputeran Berprestasi Tinggi.

Kata kunci: Penjadualan, Fungsi pemberat, Pengguna hujung

Abstract

Grid computing is a connected computing infrastructure that furnishes reliable, stable, ubiquitous and economic access to high-end computational power. The dynamic nature of the grid brings several challenges to scheduling algorithms that operate in queuing-based scheduling approach. This approach typically performs scheduling based on a certain fixed priority which leads to increase the delay for the running applications. Thus, the overall performance will be deteriorated sharply. The main aim of this study is to minimize the delay in the scheduler for the dynamic jobs. Therefore, this research tackles dynamic scheduling issues by proposing Swift Gap (SG) mechanism. SG comprises of two stages by applying two mechanisms: Best Gap (BG) and Tabu Search (TS). In the first stage, the job is placed in the earliest gap available in the local resources' schedules, while the second stage optimizes the performance by checking all available gaps among resources' schedules to find a better gap to place the job in. To further improve the performance, the Completion Time Scheme (CTS) is developed. CTS reduces the delay be placing the job in the gap that guarantees the best start time for the job, and the fastest resource available. The integration between SG and CTS (SG-CTS) is achieved by applying best start time rule in the first stage only, whereas the second stage includes both rules.SG-CTS is evaluated numerically through simulation by using real workloads that reflect a real grid system environment. The findings demonstrate that SG-CTS improves the slowdown by 27%, bounded slowdown by 25%, tardiness by 21%, waiting time by 16% and response time 7% compared to Conservative backfilling mechanism followed by Gap Search (CONS-GS). The contributions of this thesis are to assist the end user by significantly providing better Quality of Service in grid computing environment, especially for High Performance Computing applications.

Keywords: Scheduling Jobs, Weight function, End user

Acknowledgements

In the name of ALLAH, Most Gracious and Most Merciful:
“Read; And your Lord is the Most Generous, Who taught by the pen, Taught man that which he knew not;”

(The Holy Quran - Al-Alaq (96): 3-5)

Firstly, I would like to express my sincere gratitude to my advisors for the continuous support of my PhD study and related research, for their patience, motivation and guidance.

I would like to thank the heads of InterNetworks Research Laboratory (IRL) for the great support and for the massive effort in providing the facilities to conduct this study. Furthermore, I would like to thank the IRL lab-mates for their insightful comments and advantageous discussions.

I would like to express my sincere gratitude to the respected examiners for spending time and sparing no effort to evaluate this work professionally. I would like to appreciate the Research, Viva and Training Unit for their highly cooperation. I am very grateful to the staff in SOC for their guidance, assistance and kindness.

My thanks also goes to the supporting team of Alea simulator for their guidance and to the Czech National Grid Infrastructure MetaCentrum for providing the simulated workloads in this study.

Last but not the least; I would like to thank my great family, my beloved wife and son for their unique support, understanding, dedication, devotion, encouraging and endless love. Without them, I would not go far in this long journey.

Declaration Associated with This Thesis

[1] **O. Dakkak**, SA. Nor and, S. Arif, "Scheduling Jobs through Gap Filling and Optimization Techniques in Computational Grid" in Journal of Computer Science, Science Publication, pp. 105-113, Volume 13, Issue 5, May, 2017.

[2] **O. Dakkak**, SA. Nor and, S. Arif, "Scheduling the Jobs Using Backfilling Technique through Schedule-Based Approach in Grid Computing Environment for High Performance Computing Applications", UUM Website, technical report, 2017.

[3] **O. Dakkak**, SA. Nor and, S. Arif, "Scheduling through Backfilling Technique for HPC Applications in Grid Computing Environment", published in ICOS, Conference on Open Systems. Indexed in Scopus IEEE Xplore, 2016.

[4] **O. Dakkak**, SA. Nor and, S. Arif, "Proposed Algorithm for Scheduling in Computational Grid Using Backfilling and Optimization Techniques," published in Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 2016, ISSN.1843-2150.

[5] **O. Dakkak**, S. Arif, and SA. Nor, "A Critical Analysis of Simulators in Grid", Jurnal Teknologi, Volume 77, Issue 4, pp. 111-117, August, 2015.

[6] **O. Dakkak**, S. Arif and SA. Nor, "Resource Allocation Mechanisms in Computational Grid: A Survey." Asian Research Publishing Network (ARPN). Volume 10, Issue 10, pp. 6662-6667, 2015.

[7] **O. Dakkak**, SA. Nor and, S. Arif, "A Critical Review on Resource Allocation Mechanisms in Grid Computing." in The 4th International Conference on Internet Applications, Protocols and Services, 290-294, Cyberjaya, Malaysia, December, 2015.

[8] M. S. Alzboon, M. Mahmuddin and S. Arif, and **O. Dakkak**, "Peer-to-Peer Resource Discovery Mechanisms in Grid Computing: A Critical Review." In The 4th International Conference on Internet Applications, Protocols and Services, 290-294, Cyberjaya, Malaysia, December, 2015.

Table of Contents

List of Tables

List of Figures

List of Appendices

List of Abbreviations

AB	Analogical Benchmarking
API	Application Program Interfaces
BF	Best Fit
BH-PR	Biggest Hole-Priority Rule
C	Cost
CAS	Community Authorization Service
CONS	Conservative Backfilling
CONS-GS	CONS with Gap Search
CPU	Central Processing Unit
CTS	Completion Time Scheme
DAT	Direct Access Transport
DRM	Design Research Methodology
DS-I	Descriptive Study-I
DS-II	Descriptive Study-II
EASY	Extensible Argonne Scheduling System
EG-EDF	Earliest Gap- Earliest Deadline First
ERD	Earliest Release Date
F	Factor
F	Fairness
FCFS	First Come First Serve
FF	First Fit
FLEX-CONS	Flexible Conservative Backfilling
FRF	Fastest Resource First
FTP	File Transfer Protocol
GARA	General Architecture for Advanced Reservation
GIS	Grid Information Service
GRAM	Grid Resource Access and Management
GS	Gap Search
GSI	Grid Security Infrastructure
HDD	Hard Drive Disk
HPC	High Performance Computing
IDE	Integrated Development Environment
IP	Internet Protocol
JDT	Java Development Tools
LJF	Longest Job First
LM	Launching and Monitoring
LRM	Local Resource Manager
LW	Long Wide jobs
LN	Long and Narrow jobs
M1, M2	Mechanism 1, 2
MDS	Monitoring and Discovery Service
Ms	Milliseconds
MTTD	Minimum Time to Due Date
MLF	Min Load First
NUWT	Normalized User Waiting Time values
PC	Personal Computer

PS	Prescriptive Study
QoS	Quality of Service
RC	Research Clarification
RMS	Resource Management System
SFTO	Simple Fair Task Order Scheduling
SG	Swift Gap mechanism
SG-CTS	Swift Gap with completion time scheme
SJF	Shortest Job First
SLA	Service Level Agreement
SN	Short Narrow jobs
SNAP	Service Negotiation and Acquisition Protocol
SS	Swift Scheduler
SW	Short and Wide jobs
TUSA	Total User Squashed Area
TUWT	Total User Waiting Time
TS	Tabu Search
U	Utilization
URL	Uniform Resource Identifier
UWT	User Waiting Time
VO	Virtual Organizations

CHAPTER ONE
OVERVIEW

Grid computing is a set of resources, which are distributed in different geographical zones. These resources are managed through grid system. The main goal of grid computing is to make execution of massive tasks faster and within reasonable cost [1]. The importance of grid computing lies in minimizing the execution time for huge tasks for the users, reducing costs and improving the efficiency of the hardware by exploiting idle resources on demands.

In order to utilize the resources on demands, the virtualization concept is applied in the grid. The virtualization means there is no need for components which are part of that research such as database, computational power and the researcher (user) to be in one place. Thus, grid computing avoids the limitation of the resources by making the processing task time faster, due to the dedicated resources for a single task. Due to the vast diversity and high heterogeneity of the grid, it inherits many challenges in terms of Quality of Service (QoS), sharing of applications and data, compatibility of grid policies, reliability, robustness and trust [2].

According to Qureshi et al. in [3], resource allocation is assigning the suitable resource for the suitable job. Resource allocation includes scheduling and monitoring. Scheduling is the core of resource allocation. Including the scheduling itself, there are two phases, which are resources selection and job execution. Resource selection refers to the discovery of the available resource, whereas job execution refers to submission of the job to the chosen resources [3]. Due to the rapid development in data network communication technology and the existence of plenty powerful computer resources, grid computing has experienced tremendous growth and deployment [4, 5].

The emergence of grid computing has resulted the use of these resources in one single group to settle large-scale tasks in academia and research fields. These tasks belong to different fields, such as engineering and business. Grid computing provides the user with various types of applications, like tremor simulation, fiscal modeling, motion image animation, access to data repositories, visualization and data analysis and virtual classrooms [4].

Grid provides service through its infrastructure, such as the coordination for numerous kinds of resources amongst distributed areas. There are many types of resources such as storage space, processor cycles, network bandwidth, software and database. These resources could belong to various organizations and they are connected with each other via the Internet.

The technological evolution in the computing and networking fields has caused the emergence of beneficial infrastructure that is well-known as computational grid [4, 5]. Massive resources are hosted from distant places and therefore grid computing has the ability to enable users of gaining access to the distributed resources remotely [6]. Due to the high heterogeneity nature of the grid, global and local policies for different distributed computing systems have to be considered, such as: resource allocation and job scheduling, computational cycle and storage [7].

From the end user perspective, the delay is crucial for the running applications. The faster the jobs are processed, the smoother the application is. Several performance metrics that give an indicator about the delay performance such as waiting time, response time, tardiness and slowdown have to be considered. For instance, the waiting time refers to the duration of time that jobs have to wait in the system starting from the submission time until the time to be processed. While total response time is, the

aggregate time of the waiting time and the execution time for all jobs [5]. Other metrics such as the makespan and the utilization are important to be measured for a grid systems, which run for limited time [8].

This research introduces a new scheduling mechanism that improves the performance in terms of minimizing the delay for the scheduled jobs in the computational grid environment. This chapter is structured as follows. Section 1.1 presents the research motivation for this research. Section 1.2 states the statement of the problem. Sections 1.3 and 1.4 present the research questions and research objectives. Section 1.5 clarifies the research scopes. Section 1.6 highlights the significance of the study. The contribution of this study is addressed in Section 1.7. Finally, the thesis organization is outlined in Section 1.8.

1.1 Research Motivation

In scheduling and in order to schedule jobs properly, several challenges and constraints have to be met such as resource usage [9] and response time [10]. This section addresses some of the challenges and crucial issues in grid computing environment.

Resource allocation is one of the biggest challenges in the grid. There is no standard policy that a resource allocation mechanism can follow, in order to provide an optimal performance [3]. This is due to the high heterogeneity of the grid. As a result, these resource allocation mechanisms have to perform much better to meet the user's requirements. The user should not worry about the abilities and the limitations of these mechanisms.

Due to the high dynamicity of the applications on demand, most of these applications parameters keep changing frequently in terms of value or the type of the parameter.

Consequently, tasks of certain job could change frequently in a short period of time and during execution time such as real time applications. That is another challenge for these resource allocation mechanisms, which needs to be handled while the applications are running. These challenges can affect the system performance badly if not addressed duly [3].

The time spent by the jobs in the schedule is one of main challenges in scheduling, especially if the number of the jobs is overwhelming and the jobs keep arriving to the system continuously. If the operating policy that runs in the schedule failed to handle this matter wisely, the scheduler will become a bottleneck for the jobs. The delay for the jobs is mainly caused due to long waiting time in the system.

The delay factor in grid computing is quite important. Criteria such as slowdown and response time play crucial role in the level of QoS for end user. Particularly, if the grid system runs for very long time, could be up to months or years [8]. Hence, the time for the jobs that spent in the system becomes crucial. Therefore, when the delay is reduced, the QoS for the end user will be improved significantly. In this study, our motivation is to tackle the issues that are related to the job scheduling performance in real grid systems that run for quite long time. Hence, reducing the delay for the jobs to be scheduled is our utmost importance.

The scheduling mechanism that is used to schedule the jobs has a very important role in overall scheduling quality as it ultimately affects the QoS for the user [3, 8]. This is attributed to the policy that the mechanism uses, particularly when the number of the jobs is massively high. Hence the importance of the scheduling mechanism comes.

1.2 Research Problem

In order to describe the problem fully, this section is split into two subsections. The first section provides a background about the problem, while the second section states the problem straightaway.

1.2.1 Problem Background

End users send intensive complex tasks that need huge computational power to process. In order to handle these jobs, the users send their tasks to the resources, which are dedicated for tackling such kind of jobs. From the perspective of the end user, getting the job done swiftly is highly required. Thus, QoS becomes very crucial and important to achieve [3, 11, 12].

There are two main types of scheduling algorithms in order to schedule the jobs. The first type is the classical algorithms, whereas the second type is the backfilling algorithms. The classical scheduling algorithms use the queues system approach; this approach does not require any kind of information from the arrived jobs. The main problem in the queues system is that it cannot plan the scheduling for the next arriving jobs, which may lead for long waiting times for these jobs in the queue [11].

Thus, classical scheduling mechanism that applies queues approach are very efficient for low number of the jobs. To solve this problem, a new approach was devised. This new approach is called planning scheduling or schedule-based system. In this type of scheduling, the mechanism will be implemented in a "scheduler." The scheduler is a data structure, which requires specific information about the incoming jobs in order to schedule them. Hence, the mechanisms have the ability to set the scheduling in advance [13]. In this approach, the backfilling technique is required to block the gaps

without specific order.

As the time is running in a dynamic grid environment, the newly arrived jobs reach the schedule, meanwhile some of the jobs are already processed [11, 14]. In addition, the size of the jobs differs from one job to another. Moreover, the arrival times and the submission times for these jobs are not the same. In the interest of time, this would create gaps in the schedule. These gaps are empty spaces in the schedule.

The processor stays idle, while these gaps take time in the schedule. Thus, the processor will be wasted for awhile especially when the number of gaps is increased due to thousands of jobs that reach the system. Obviously, the previous inefficiency of the system will cause a delay, which in turns affects the QoS for the end user.

As a result, backfilling is required to extremely exploit the resources. Backfilling was described as “a something for nothing, a benefit without a tradeoff” [15]. Some backfilling mechanisms such as Conservative Backfilling (CONS) and Extensible Argonne Scheduling System (EASY) have been proposed by previous researchers in order to improve the QoS. The proposed mechanisms had improved the QoS for the end user by utilizing the resources. However, these mechanisms still have some weaknesses, such as delaying the subsequent jobs due to the aggressive backfilling, or less system efficiency due to conservative approach in backfilling.

1.2.2 Problem Statement

The classical scheduling mechanisms schedule the jobs based on certain order. These mechanisms perform very well for small grid system [16], while they suffer when the grid system is large and massive number of jobs need to be scheduled. This is due to the lack of utilization of the available resources caused by the inflexible approach

applied by these mechanisms. Lack of utilization will lead to deterioration the QoS for the end user since the jobs will take longer time to be scheduled [11]. Moreover, this approach cannot strike a balance among the performance metrics because of the applied fixed policy [17-19].

Backfilling mechanisms has better utilization of the resources, hence they perform much better than classical scheduling mechanisms. There are two well-known backfilling mechanisms, which are EASY and CONS. EASY [20] is a robust and aggressive backfilling mechanism. However, this mechanism still suffers from some weaknesses, such as delaying the subsequent jobs, which are still ahead of the candidate backfilling job. This happens when the aggressive backfilling for jobs is conducted. Thus, other jobs ahead could be delayed [14].

CONS [21] checks if the backfilling for each job will cause a delay for the jobs ahead or not, thus the jobs can be backfilled as long as they fit in the gaps without causing any delay for the rest of the jobs. A serious concern would be that following this restrictive approach would perform less efficient scheduling compared to the aggressive approach [15]. The applied approach in CONS will lead to less number of the backfilled jobs. Thus, CONS avoids the weakness of EASY, but it raises other problems related to the system efficiency and more waiting time for the jobs for backfilling since each job must get a reservation first.

Other mechanisms, proposed the optimization solution to overcome the issues that CONS suffer from [11]. However, this solution ignored the importance rule of job's completion time by focusing on other criteria such as makespan and the number of delayed jobs, which are not crucial and significant for the QoS for the end user. Furthermore, the aforementioned criteria are not suitable for massive grid system that

runs for a quite long time [8].

In addition, the backfilling mechanisms always try to place the job in the earliest gap. The earliest gap that selected, does not guarantee the best finishing time for the job when the resources have different computational power [22]. Moreover, backfilling mechanisms utilize the resources fully by applying classical utilization. Classical utilization improves the efficiency of system. However, classical utilization may not be the best choice for the job to be backfilled when this concept is always applied at the expense of the faster machine selection [23].

Thus, a mechanism for a real grid computing system that concerns about QoS for the end user needs to be developed. This will be achieved by applying multi-level scheduling [5, 12] in order to exploit the available resources and concerning about job's completion time to further minimize the delay for the jobs in the schedule.

1.3 Research Questions

In addressing scheduling performance issues in grid computing environment, the following research questions were raised up:

I. How to increase the number of the backfilled jobs in order to improve the system efficiency without delaying the rest of the jobs ahead in the schedule for real grid system?

II. How to avoid the defect in the scheduling performance when the earliest gap and the classical utilization concepts are always performed?

III. How to produce a scheduling mechanism that concerns about both

aforementioned questions?

1.4 Research Objectives

The aim of this research is to improve the performance in the computational grid related to the QoS for the end user by proposing Swift Gap with Completion Time Scheme mechanism (SG-CTS) that reduces the delay for the jobs in the schedule and optimizes the performance of the scheduling. This aim can be further explained by the following specific research objectives:

I. To design a scheduling mechanism that increases the efficiency of resources and makes a reservation for each backfilled job by applying optimization technique preceded by a backfilling approach that reserves the idle time for the resources.

II. To develop a completion time scheme that evaluates the backfilling decision for each job using the weight function based on best start time and the faster machine.

III. To integrate the proposed completion time scheme with the designed scheduling mechanism by considering the features of the backfilling and the optimization mechanisms in order to maximize the scheduling performance.

1.5 Research Scope

Grid computing provides decent service level for applications that demand high computational power. Scheduling has crucial role in mapping the resource to the grid job. Grid consists of several layers, which are; fabric layer, connectivity layer, resource layer, collective layer and application layer. This research focuses on Collective Layer

since this layer is responsible for monitoring, resource discovery, brokering service and scheduling. Scheduling is the main focus of this research. The scheduling concerns about the hardware resources, whereas the considered resource type is the Central Processing Unit (CPU). Finally, the proposed mechanism is aimed to improve the performance for High Performance Computing (HPC) applications. Figure 1.1 shows the scope of this study. The bold aspects inside boxes in the diagram refer to the scope of this research.

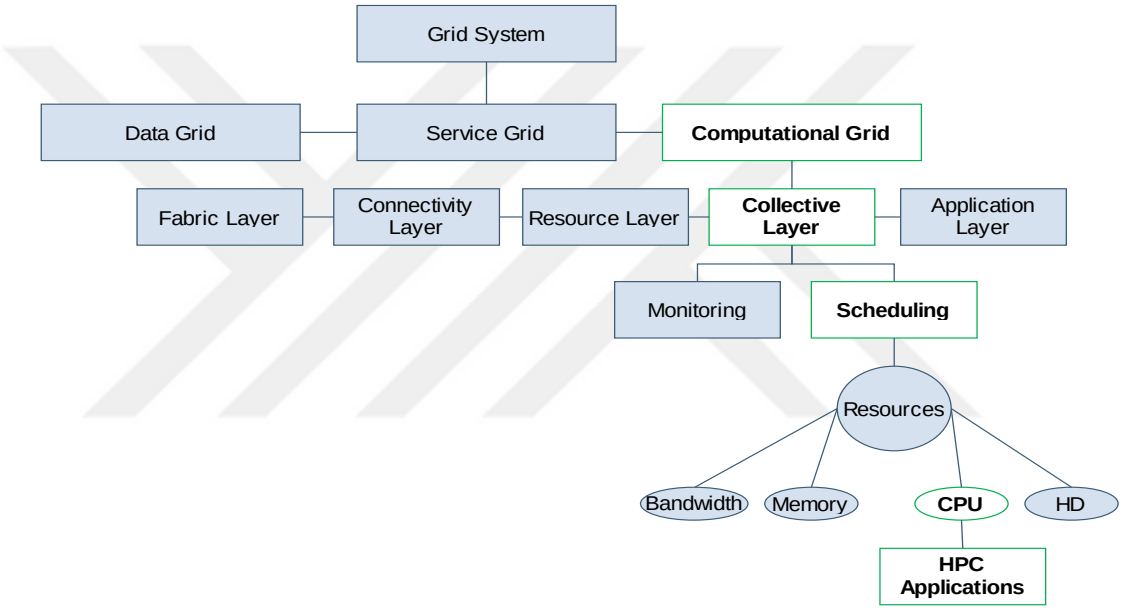


Figure 1.1. Research scope

1.6 Significance of the Research

The main task of the grid is to allow users to access the distributed resources by handling the heterogeneous resources properly. Managing such environment, especially when jobs arrive dynamically is very complicated, particularly if certain criteria have to be met. In the grid, several applicants are non-tolerant when it is related to the QoS. Many applications are very sensitive to delay. In addition, most of the grid system business models are based on allocating the resource for the job for a certain

time, if the job processing is not over, the job has to wait until the next cycle. For this, reducing the delay in the scheduler, in order to process and submit the jobs successfully to the end user is required. Thus, the need for scheduling mechanism, which minimizes the delay for the jobs, subsequently improving the QoS significantly becomes a necessity.

1.7 Contribution of the Research

The contribution of this study is to come up with a new mechanism, named Swift Gap (SG). This mechanism exploits the feature of planning scheduling approach. SG is the outcome of the integration between Best Gap mechanism and Tabu Search (TS) using high hybridization level. SG improves the QoS for the end user significantly. Later on, this study aims to develop the Completion Time Scheme (CTS), the aim of this rule is to further minimize the delay. The developed CTS is going to be applied in SG mechanism. The total outcome mechanism is SG with Completion Time Scheme (SG-CTS), which achieves high QoS level for the end user.

1.8 Organization of the Thesis

This thesis consists of seven chapters, which are organized as the following:

Chapter One: This chapter gives an overview about the area of the research, which also includes research motivation and introduces the problem statement. This chapter also addresses the research questions and research objectives, presents the scope, the steps, the contribution and the significance of the study.

Chapter Two: This chapter covers the literature review from the previous and current related work. Moreover, this chapter presents relevant information for better

understanding of the selected research area.

Chapter Three: This chapter provides the details of the selected methodology that is used for this study.

Chapter Four: This chapter presents the SG mechanism. In addition, it addresses the approach that SG is based on. Moreover, a verification and validation for the introduced mechanism is provided.

Chapter Five: This chapter introduces the Completion Time Scheme and its impact compared to Swift Gap without this scheme.

Chapter Six: This chapter presents Swift Gap with Completion Time Scheme. The integration of the Completion Time Scheme with Swift Gap in order to obtain high performance is explained. In addition, an extensive evaluation of the final mechanism is introduced.

Chapter Seven: This chapter concludes the study, highlights the contributions in the body of knowledge, recommends for the future work and shows the limitation of this study.

CHAPTER TWO
LITERATURE REVIEW

This chapter presents a comprehensive literature review of previous related works to the conducted research. It presents the concept of grid computing in Section 2.1 in order to put this research in its context. A comparison between queue scheduling against the planned scheduling is provided in Section 2.2. Section 2.3 addresses Service Level Agreement concept (SLA) and how it improves the Quality of Service (QoS) in grid environment. A comprehensive review of the current related works is presented in Section 2.4. This section provides critical review about the existing mechanisms that are related to this study. Section 2.5 briefly reviews Tabu Search (TS) approach. Finally, this chapter is concluded in Section 2.6.

2.1 The Concept of Grid Computing

Grid computing is computational technology which aims to get the maximum benefits from idle resources, such as CPU cycles, memory, bandwidth, storage and so on.

The main idea behind this technology is to connect these idle resources together to one network, thus a virtual system will be created in order to share and manage the resources dynamically during operating time. Through the grid network, the grid system can supply sophisticated quality level services and access to a massive number of distant resources to any user at any time. Unlike the web, which uses Internet Protocol (IP) to gain access to any content on the Internet via Uniform Resource Identifier (URL), Grid computing needs to have access to computational resources consistently [2, 4].

Users are able to use computing resources like database, hardware resources for many

various devices that are diffused everywhere, via very massive virtual network known as "Grid Computing ". For instance, let us assume that we have 20 computers and half of these Personal Computers (PCs) are busy while the other PCs are idle. The key idea is to use the CPU cycle for these idle PCs, as well as the possibility to use some or all of the other PCs busy CPU, in case that these PCs are not using the whole cycle of their CPU and unify all the aggregate of processing power to handle a huge task. Figure 2.1 presents the concept of grid computing.

Figure 2.1. The concept of grid computing

In the middle of 1990s, grid began to deal with massive-scale computing problems through resource sharing available on the network. This assistance from the grid at that time was only including getting assistance from dedicated supercomputers to participate in solving computational tasks. Supercomputer is very costly and it is difficult to get access to. Thus, moving to cheaper resources was the key motivation to go for associate resources that include computational, storage and bandwidth which are distributed geographically in numerous places.

Existing resources with their hardware, operating systems, local resource management

and security infrastructure are the main concern for the grid. Virtual Organizations (VO) are a logical entity within dispersed resources can be located and shared as if they came from the same organization. To support virtual organizations, grids identify and give a set of standard protocols, middleware, toolkits and services built on the above of these protocols. The main focus for grid infrastructure are security and interoperability due to resources could reach to various administrative domains locations that have both global and local resource employment policies, dissimilar hardware and software options and platforms differing in accessibility [3].

Five layers form the main components of grid protocol architecture. In turn, grids offer protocols as well as services to these layers. The first layer is the fabric which offers access to various resources such as computing elements, storage and bandwidth. Grid depends on available fabric parts such as Local Resource Manager (LRM) to provide the resources, for example, the general-purpose components such as the General Architecture for Advanced Reservation (GARA) [24] and specialized resource management services like Falkon [25, 26].

For smooth and safe network transactions, core communication and authentication protocols are needed. These protocols are provided by the connectivity layer [27]. The Grid Security Infrastructure (GSI) [28] protocol lies behind every grid transaction. Discovery, negotiation, monitoring, accounting and charging for sharing operations as well as exclusive resources are the responsibilities of the protocols which are identified by the resource layer [26]. Grid Resource Access and Management (GRAM) [29] is responsible for allocation of computational resources, handling the computation for those resources and monitoring. In addition, Grid File Transfer Protocol (GridFTP) [30] provides data access and for high speed data transfer.

The Collective layer is a key-layer in this architecture. It is responsible for coordinating interactions across directory services such as Monitoring and Discovery Service (MDS) [31], co-allocation, scheduling, brokering services such as Condor-G and Nimrod-G [32, 33] and Community Authorization Service (CAS) [34] for global resource policies and data replication services [4].

Finally, the application layer includes all user software, which is developed above protocols and Application Program Interfaces (API). The application layer runs in the VO environment. Figure 2.2 shows the interactions among grid layers. Figure 2.3 presents the conceptual model of the grid.

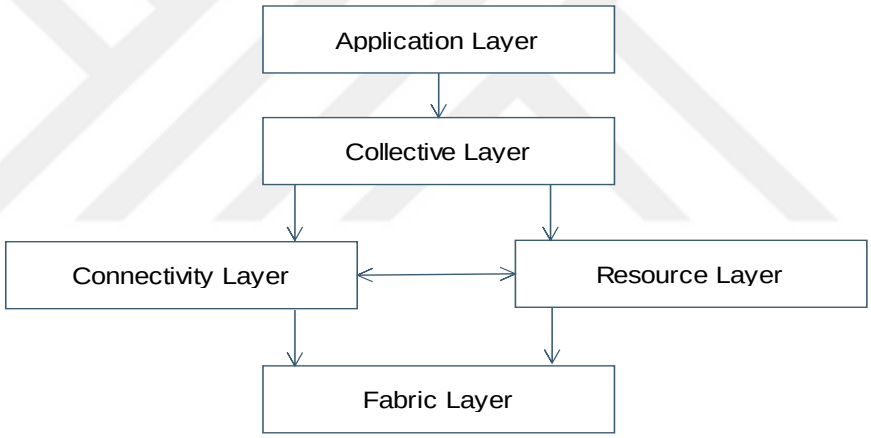


Figure 2.2. Grid layers interactions

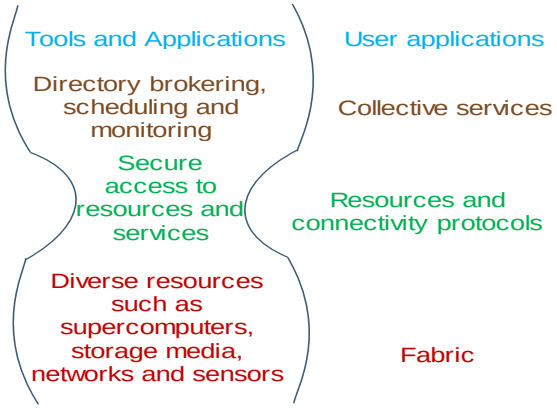


Figure 2.3. Conceptual model of grid [35]

2.1.1 Grid Working Steps

The resources in a grid system are the main key to offer the services to the user. To get these resources, first the grid network has to discover the resources and then the collective layer will schedule the job which is requested by the user. Grid scheduler can exist in several approaches in real network scenario. These approaches are: hierarchical, centralized, decentralized [36] as well as peer to peer. To establish request-to-resource mapping, the grid scheduler needs to know the resources condition. This information is very significant to complete the mapping between the request and the resource. Grid Information Service (GIS) [37] is responsible to provide the required information. This information about resources include CPU cycles, data storage availability, application availability and the available bandwidth. In addition, before assigning resources for the requested grid job, Analogical Benchmarking (AB) tests the quality of the resources performance [38, 39] in order to decide if the resource is suitable for executing the job or not.

Relying on the information that comes from GIS and AB, the grid scheduler chooses the group of resources and time to run these jobs. Delivering grid application jobs to the chosen resources, arranging input data that come from managers if needed and monitoring the performance of the applications are the responsibilities of Launching and Monitoring (LM). Grid Resource Allocation and Management (GRAM) is one example of LM [40]. Local Resource Manager (LRM) is responsible for local scheduling within the resource domain and informing about resource information to GIS. Figure 2.4 presents how grid system works as previously discussed.

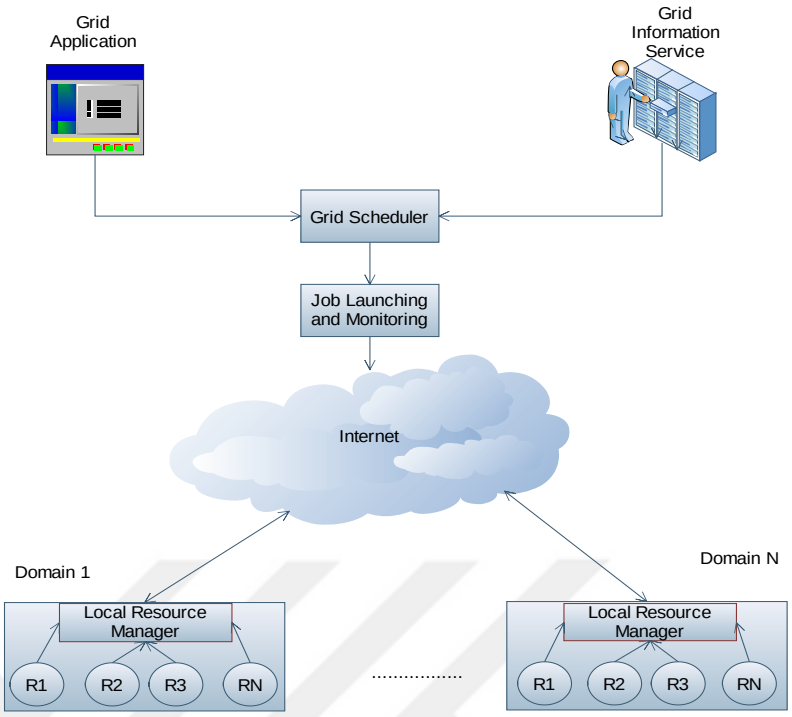


Figure 2.4. Grid scheduling process [37]

2.1.2 Resource Allocation Concept

While the grid offers a myriad of services, resource allocation can be considered as the main and basic service that responds to the service requester [41]. Resource allocation refers to assignment of the suitable resource to the suitable job. Wu et al. in [42] has defined resource allocation through a mathematical equation.

Based on [42], when a job is allocated to any types of resource, this will generate two measurable values which are: Utilization, (U) and Cost, (C). Utilization refers to the amount of benefit that user reaps from the resource. Utilization also can be represented as the amount of completed processed tasks. Cost can be considered a value that is generated from the use of that resource such as latency, high bandwidth, charge for using the resources and so on. If resource allocation is the main objective O, therefore, the given equation will be:

$$O = U - C \quad (2.1)$$

Where O is the optimal resource allocation, U is the utilization and C is the cost. From this equation, it is clear that the bigger the utilization, the better optimal resource allocation will be. In other hands, the smaller the cost, the better optimal resource allocation. In case that the cost is high and the utilization is small, the quality of the resource allocation that is offered to the user will deteriorate. To attain optimal resource allocation, the mechanism has to maximize U and minimize C. In resource allocation, the resource may be continuous value or discrete value. The continued value can represent a resource value when the value is not negative and continues at the same time. Processor time is an example of a continuous value. The resource may take discrete value, when the value is not negative and indivisible, such as person, car and so on.

The main aim of resource allocation mechanisms is to guarantee an acceptable level of the QoS to the application for user requirements [41]. These mechanisms are taking a significant role when assigning the resources for the user's job. The mechanism could be static or dynamic. The static mechanism type allocates the resources based on the compile time for the application, while the dynamic mechanism allocates resources based on run time. Static mechanisms are more stable and their behaviours are easy to predict. Dynamic mechanisms, on the other hand are required in a dynamic environment such as the grid [41].

2.2 Queue Scheduling and Planning Scheduling in HPC Systems

High Performance Computing systems (HPC) are managed by resource management systems. Previously, resource management system is most likely operated based on queuing approach. However, with the increasing number of jobs and high

heterogeneity level in the resources, the queue approach suffers a setback to provide high QoS because the queue approach cannot provide advanced reservation for the jobs. Conversely, planning scheduling approach (schedule-based) has the ability to manage the resources for current and future time and that improves the quality of scheduling for applications that run in HPC systems.

2.2.1 Queue System and Planning Scheduling Structure

The scheduling process in the queues consists of two stages. The first stage orders the jobs based on the applied policy. This is achieved by the queue management. The second stage applies the scheduling policy. Generally, it selects the job from the queue and allocates it to the available resource(s). Job allocation is straightforward when only one resource is required to be allocated. However, when several resources meet the job requirements, node allocation process will be applied in order to decide which resources will be allocated to.

First Fit (FF) policy assigns the job to the first available resources that can execute the job. Best Fit (BF) allocates the job to the minimum number of resources that can tackle the job. This will help in saving the more powerful resources to handle larger jobs in the future. On the other hand, Min Load First (MLF) allocates the most powerful resources to the job, even though if the job does not require such huge computational power. Fastest Resource First (FRF) selects the fastest available resource [43].

Queue system performs the scheduling when the resource is available and the job is ready to be executed. Thus, this approach is suitable for “the present time” only since the applied scheduling solution is applied at the last possible moment. This explains why optimizing the solution later is not possible in this approach. The optimization is

doable only when the scheduling policy is completely redesigned. Figure 2.5 presents the structure of this approach.

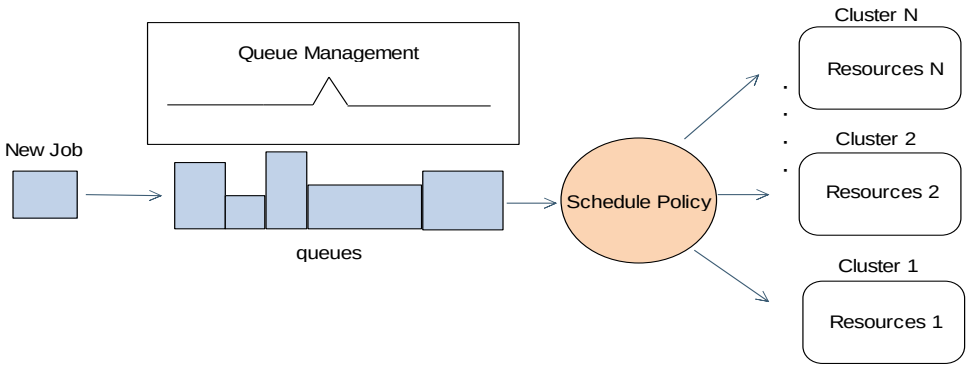


Figure 2.5. The structure of queue system

Planning approach handles the current time and the future events of the system. This is due to its structural design that represents the resources and system time as (X, Y) axes instead of stored jobs in queueing system. The axis X represents the resources, whereas Y represents the time system. Since planning approach concerns about current and future state of the system, non-trivial scheduling decision must be taken for every job that reaches the system, and this is the main difference between queue and planning approach. Figure 2.6 presents the structure of this approach.

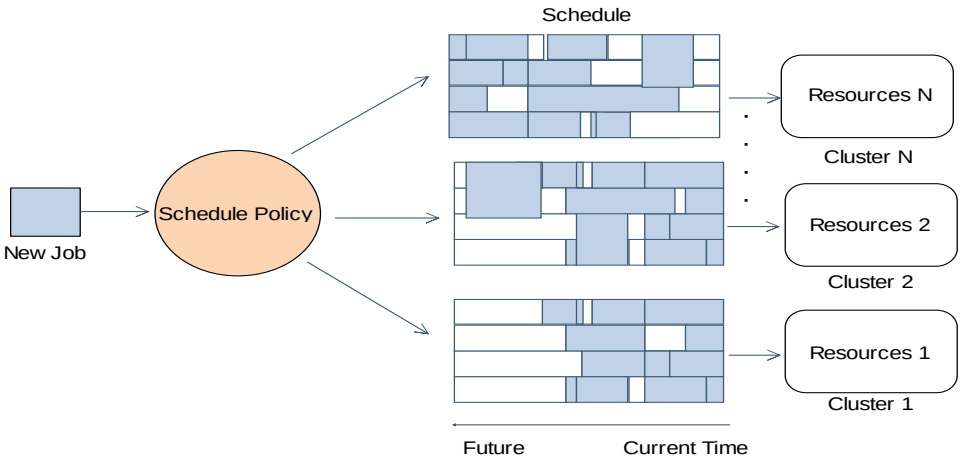


Figure 2.6. The structure of planning system

The following section comprehensively reviews both approaches and provides a comparison between them. However, before going into this comparison, the concept of backfilling has to be described, since the concept of backfilling is implicitly involved in Section 2.2.3 and Section 2.2.4.

2.2.2 The Concept of Backfilling

In distributed systems, resources can be free of charge or paid. The paid resources cost the administrators extravagant expenses in order to provide the power and the cooling system to maintain the resources so that service is constantly available. Thus, utilizing these resources fully is extremely important to satisfy the users needs and to reduce the huge expenses. Classical algorithms suffer in terms of utilization due to the fixed order in handling the jobs. Hence, the algorithms that use backfilling concept were proposed. The backfilling has no fixed order to execute the jobs. For instance, if J2 is a head in the queue and there is no sufficient resource to handle it, a job J3, which is behind job J2 in the queue or scheduler can be executed if the same resource has the ability to handle it. Thus, the idle time for the resource is invested rather than being wasted [15].

The experiments in [44] have shown that the backfilling concept improved the total utilization by 20% in most systems and workloads. Subsequently, the response time for the jobs is improved. The experiments also have shown that 90% of short jobs in the workload can be backfilled. Moreover, it was observed that even the largest jobs have a better chance to start earlier since the short jobs are executed faster and thus the largest jobs step forward in the queue or schedule faster [15].

Backfilling is the upgraded version of variable partitioning framework. In variable

partitioning, users specifies the number of requested resources and the duration of the jobs to be executed. The variable partitioning refers to the created space in the processer in order to be allocated based on the job requirements. The number of the requested resources has to be fixed during the execution time in ideal scenario. Thus, the requested jobs can be described as a relationship between the processors and the time.

The backfilling will be conducted based on the information provided by the user or based on historical memory of the request. Based on the user provided information, the system identifies the gaps before moving the job into the suitable gap. This is the core of backfilling, where the main aim is to move as many short jobs forward as possible in order to improve the system efficiency and to reduce the delay.

2.2.3 Queue Scheduling Approach

In the queue system, the requests of the jobs are executed based on a certain order. The job that has the highest priority in the order; such as Shortest Job First (SJF) and First Come First Serve (FCFS) will be positioned ahead in the queue and will be executed first. Queue system allocates the waiting jobs to the free resources. For instance if there are surplus resources then another queue can be established. Usually, the extra resources that allocated for the newly established queue are less powerful from the resources in original queue. If there are no extra resources, only one queue exists and the jobs in the back of the queue has to wait until enough resources become available while the jobs that have the highest prioritized are being processed.

The extra resources in the queues, which are allocated to the newly established queue, are activated at certain times such as weekend or prime time. The least prioritized jobs

can be allocated to the extra “idle” resources using backfilling mechanisms, such as Extensible Argonne Scheduling System (EASY). In queues, the information about the number of incoming jobs and the resources capabilities are not required unless if a backfilling policy is applied.

The queue scheduling approach is not originally designed to require information to backfill the jobs, such as when the peak time will be or when the start time for the job is going to be. Thus, the process to make the reservation for the jobs for backfilling purpose becomes bothersome and tiresome. Nevertheless, queue scheduling approach is still widely used, especially for small grid systems due to its simplicity and the fast scheduling decisions [13].

2.2.4 Planning Scheduling Approach

In this approach, the scheduling is set in advance by reserving the resources for both the present time and the future time as well. The scheduler has to be aware about the arrival time of the incoming jobs as well as the execution time besides the information about the available resources. The prior knowledge is obtained from users or can be predicted based on the historical memory for the running jobs for a specific workload [13].

Once the scheduler is fully aware about the incoming requests, the planning becomes possible. Of course, there are no queues in this approach. All incoming jobs will take a place in the scheduler by reserving the suitable resources based on the computational power ability required for each job. When the job execution is over, or when a new job reaches the system, the scheduler has to be updated in order to allocate the resources for the job according to the new situation of the scheduler. The jobs that cannot reserve

resources will be at the back of the scheduler and will be scheduled based on a certain policy, for instance FCFS.

It can be observed that a backfilling is implicitly carried out during the planning and re-planning process. The backfilled jobs may jump ahead while previously existed jobs may move behind the backfilled jobs. Delaying or non-delaying the previously existed jobs depends on the policy of the applied backfilling algorithm [13].

Definitely, this approach requires extra computational time to compute the scheduler and to update it for the newly arrival jobs. Thus, the requests in the queue approach are faster than planning approach. In addition, the users have the ability to view the scheduler and that may bring cause complaints by them such as “my job can start earlier” or “my job can fit here”.

2.2.4.1 Planning Scheduling Functions

In the following sections, some features are explained. These features can be implemented without the planning approach. However, the design of this approach promotes them.

2.2.4.1.1 Requesting the Resources

Requesting the resources is the responsibility of Resource Management System (RMS). When the user requests for resources or sends a job, RMS responds based on a certain attribute, such as the duration, number of desired resources and required memory. Usually, this is achieved even if the user specifies the desired resources exactly or submits the minimum requirements to handle the request. To achieve this, RMS has to deploy the requests and negotiate for the resources [13]. Deploying the

requests can be done by submitting a range of the required resources to handle the job. This range is specified by the user. The other way is to use the “optimizer”. This component assists RMS to allocate the resources based on one or more attribute. The optimizer has to know about the properties of the incoming job and the desirable objective function from the user, in order to select the right resource. Otherwise, the optimizer will select the resources by default [13].

The co-allocation in grid environment allows the user to co-allocate the resources. Co-allocation means several and various types of resources can be reserved at the same time. However, this is not simple procedure, since there is a possibility that not all requested resources are available at the same time with the required quality and quantity. Deploying the requests eases this task. However, a negotiation protocol is required to complete this procedure, for instance the Service Negotiation and Acquisition Protocol (SNAP) [45]. Figure 2.7 [13] illustrates the aspect of deploying requests. For example, to diffuse a request to reserve a resources, the flow control is (1,4). To reserve a resources using the optimizer component, the flow control is (1,2,3,4). In order to negotiate about the resources, the path has to be (1,2,3,6,1,…).

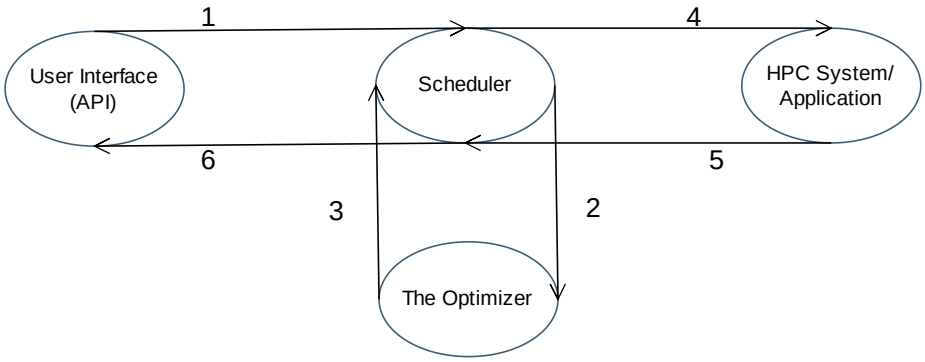


Figure 2.7. Deploying requests function in RMS

2.2.4.1.2 Dynamic Aspects

Due to the high heterogeneity of HPC systems, diffusing servers that have the ability to operate several systems are required in order to utilize the resources fully. These servers match the application with the requested resource. All these dynamic aspects are managed by RMS. The following sections present some of these aspects.

a. Variable Reservations: In the regular resource, reservation includes information about the start time for the requested job, the duration of the job and number of requested resources. The start time can be stated directly or implicitly maybe given by providing the end time for the request. The start time is fixed; the resource reservation is submitted based on the fixed time for the start time.

In case that the requested resource is free and the request is needed to be planned before the fixed time, the resource reservation will not automatically match the new submission time for the request. Thus, a new variable (V) has to be included. This variable notifies the user whenever the resource is available [13].

b. Resource Reclaiming: In HPC systems, space-slicing policy is widely applied. This policy is usually applied for parallel jobs that require dedicated resources. The so-called evolving applications [46] have several stages during the executions, such as computation, communication and checkpointing. Basically, the need of multi-types of resources is supported by RMS as discussed in Section 2.2.3.1.1.The application should be able to communicate with RMS through the Application Program Interference (API) in order to ask for extra resources or for releasing any resource after the runtime is over [13].

HPC systems have many communications networks and these networks are integrated

with each other through the software layer, for instance Direct Access Transport (DAT) [47, 48]. Thus, the application can switch from network type to another type or from low bandwidth to high bandwidth according to its needs. The path is based on Figure 2.7 where in order to deploy the path will be (5,2,3,4,5…).

DAT protects the application from network failure by implementing a failover mechanism. RMS has to be able to switch the application for a while, to another lower resource power in case the current application does not need the current high power temporarily, while another application does. This procedure helps in improving the total utilization for the system and contributes in making the requests finish according to their deadlines [13].

c. Automatic Duration Extension: This approach has the ability to enable the users to extend the runtime of their current running jobs. The flexibility in extending the runtime is very important due to the possibility of aborting the current running jobs if the specified running time is over while the processing stage for the jobs is still ongoing. This issue is well known and the repercussions are costly [21, 49] because aborting jobs without getting the desired output will be a huge waste for both user time and the power of the resources.

To avoid this, users usually tend to multiply the expected runtime by a substantial factor [21, 49]. Even though this may delay the jobs ahead, this is still much better than aborting jobs which are about to finish. The reservations for the jobs ahead stay as it is. The only matter that changes is the start time for the subsequent jobs. It is worth noting that a little delay for these jobs is better than resubmitting jobs that aborted, which in turns consumes time and resources to redo the whole process [13]. Nevertheless, the delay for the subsequent jobs that resulted from extending the

runtime jobs can be solved by applying the compression function.

d. Sharing the Space: Even though backfilling technique tries to utilize fully the resources by filling the gaps with short jobs that can fit in these gaps, there still some gaps that no job can fit in them. An application that runs for long time, but the result of that application is not urgently needed, thus, this application can utilize this gap. This technique also called “cycle stealing”, whereas the scheduler runs the application in the background.

Usually, an interrupted application restarts itself in order to utilize this feature, since the resources are reserved for the new jobs. A user has to notify the RMS about this special request and the requested resources by deployment server. Then RMS can optimize the requests using the optimizer component in order to arrange the idle gaps in the resources for applications that require to be processed in the background.

For example, if there are 15 resource available and there are two applications (A and B) that demand to run using this feature. Application (A) needs for eight resources while application (B) needs 4-32 resources. RMS allocates eight resources for A and seven for B. Table 2.1 summarizes the differences between queue scheduling approach and planning scheduling approach.

Table 2.1
Differences between queue approach and planning approach

Aspects	Queue System	Planning System
Considered Time Frame	Present Only	Present and Future
Job Submission	Into Queues	Scheduler
Proposed Start Time	No	Yes (all requests)
Runtime Estimates	Optional	Needed
Resource Reservations	Not Supported	Supported

Table 2.1 continued.

Backfilling	Optional	Mandatory
Strengths	Fast Scheduling Decisions, Simple, Good for Small Grid Systems	High Resource Utilization, Suitable for Real Grid System, Several New Features.
Weaknesses	Low Resource Utilization, Poor Performance in Real Grid System	Sophisticated, Extra Computational Time for Reservations
Examples	PBS, NQE	CCS, Maui, MetaCentrum

2.3 Service Level Agreement (SLA)

Due to the emergence of network applications that require a reservation of the resources during their lifetime [50], classical approach such as best effort is no longer enough to satisfy the customers. The computing environment has to provide the dedicated resources such as the processer, network bandwidth and the memory in order to fulfil the QoS.

This can be achieved by enabling the users to utilize the resources for a limited time by reserving the suitable resources for the demanded task. This is named as "Advanced Reservation" [24]. The advanced reservation was defined as a negotiation between the user and the resource owner in order to dedicate a particular amount of the resources for a specified time, to handle an application being run by the user [51].

Even though advanced reservation provides the application with the requested resources within agreed plan, this is still not enough in real world. The user in the real world does not care about the limitation of the resources in advanced reservation approach caused by technical reasons [52]. Hence, the concept of SLA comes in [53]. SLA is a business relationship between the user and the service provider. In this

relationship, every party has its own obligations and expectations [54]. SLA covers many other aspects such as; the scope of the agreement, penalties and limitations [55].

Since the high complexity in analyzing the rules in order to make sure that SLA satisfies both user and service provider, the SLA has to be managed automatically to match the running application with the right performance metric. Hence, the priority criteria are vary from SLA to another, depending on the running application.

The following example clarifies how SLA looks. A computing center commits to fulfill any job at maximum requires four clusters, 12 hours per day for one month. It can be noticed here that SLA offers an option, not a limited resources that matches a particular application. SLA does not consider any technical issues for the reservation such as failure in network or insufficient resources. SLA bypasses the scope of resources reservation and simplifies the agreement between the user and the resource provider. The SLA between any two parties has to be considered in the scheduling process and during the runtime for the application.

2.3.1 SLA-Aware Scheduler

From the scheduler perspective, the part of SLA starts with the negotiation processes. The scheduler has to ensure that all resources that intended to be allocated to the user based on the SLA agreement between both parties, are available. During the running time, the scheduler is concern about submitting the granted resources only, while accomplishing all the clauses successfully according to the SLA agreement is out of the scheduler's scope.

In case that a resource or a part from resource hardware had a failure, RMS has to deal with situation and that is very crucial to SLA-aware scheduler. For example, when

failure happened, the affected jobs have to be rescheduled to other resources that can fulfil the SLA agreement. If there is no available resources, the SLA-aware scheduler will delay or cancel the scheduled jobs, or even terminate the current running jobs in order to maintain the fulfilment alive. Since the attributes of SLA agreement have to be considered during the scheduling process, a simple policy such as FCFS cannot be implemented.

2.3.2 Forwarding the Job through the Grid

Even though scheduler is ready to deal with sudden situations, such as resource failure, by rescheduling the job to another resource that has the capability to fulfil the job's SLA. If there is no available resource that can handle the job based on the SLA for the job, the scheduler will violate the SLA, since SLA for the jobs are not based on the best effort approach.

To avoid any violation in the SLA for any job while the system is connected to the grid, the job will be forwarded to the grid. The scheduler will match the job to any resource in the grid that can fulfil the job's SLA. This can be achieved by forwarding the request to a grid resource broker. The grid resource broker assures the matched resource is able to successfully fulfil the SLA for the job by selecting a resource that meets the requirements requested in SLA. However, the scheduler may violate the SLA agreement and pay the penalty in case hiring a resource from the grid is economically high and will cost more than the total income when all SLA's clauses are achieved.

2.4 Related Work

This section reviews comprehensively some of related work in scheduling mechanisms in grid computing environment. Since, the proposed mechanism in this study is based on backfilling and optimization techniques, the presented review highlights the mechanisms that share the same aspects and techniques.

2.4.1 Conservative Backfilling (CONS)

CONS is one of the backfilling mechanisms [14], that applies restrictive approach in order to guarantee that the backfilled job will not delay the jobs that are ahead in the schedule. Once a short job reach the system, CONS starts looking for enough number of processors that can handle the nominated job for backfilling. Once the sufficient number of processors is located, a checkpoint namely “anchor point” is set. The anchor point refers to the time when minimum processors are available to backfill the job.

From this point, CONS examines all the selected processors’ schedule to make sure that all their future time lines have no conflict with the total running time of the nominated job. If one resource has a conflict in its future time line with the running time of the selected job for backfilling, the job will not be backfilled. Then, CONS has to search again for another anchor point. CONS keeps the scheduler updated to get new information about the current state for processors. Once the job’s anchor points becomes the current time, the job executions starts. Figure 2.8 illustrates how CONS mechanism works.

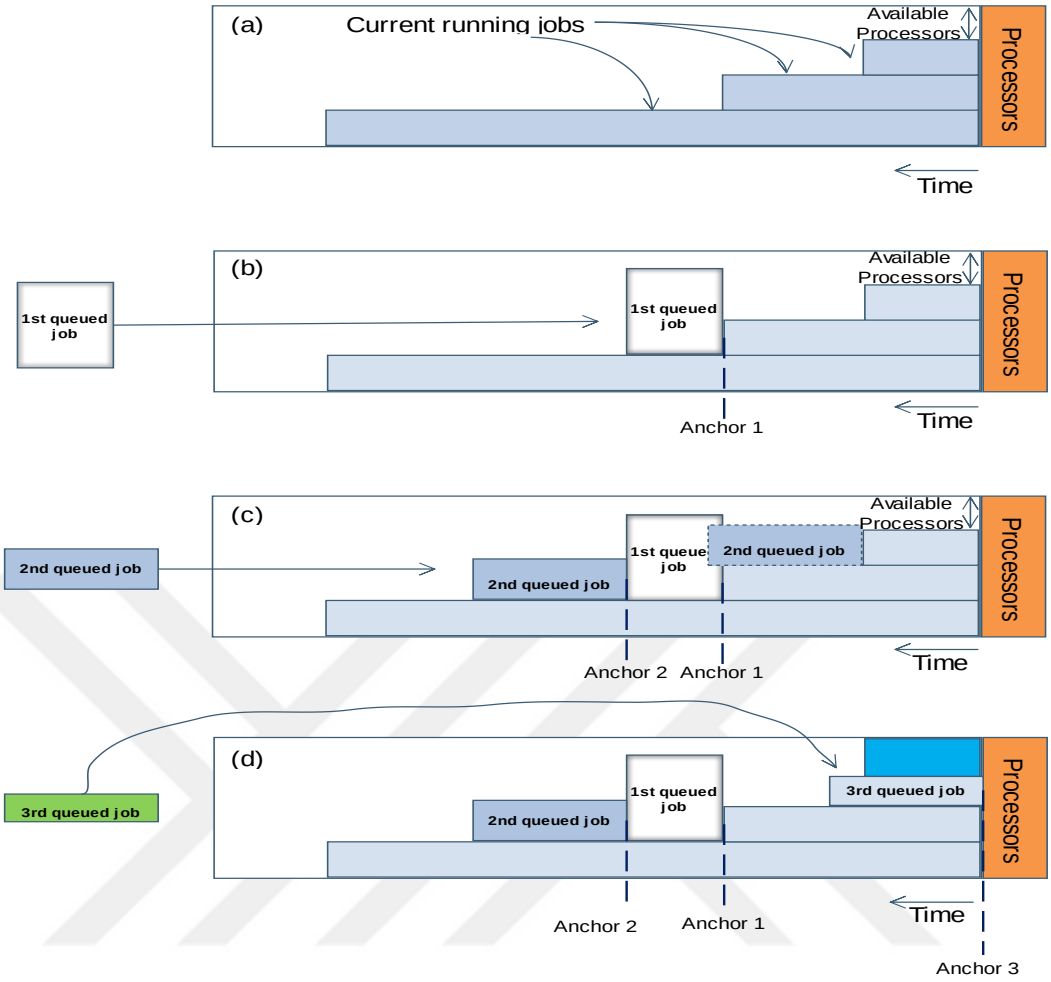


Figure 2.8. Conservative backfilling example

From Figure 2.8, it can be observed that the second queued job was not backfilled at the expense of any job. In case of (b), at the moment when anchor one starts, the second queued job was tested for backfilling. It was detected that the second queue job will interfere with the first queued job if the second queued job would be backfilled. Consequently, this will lead to the first queued job being delayed. Hence, CONS did not backfill the second queued job. In the last case, case (d), the third queue job can be backfilled without delaying any other job, thus it was backfilled immediately. It is important to note, that anchor one, anchor two and anchor three are numbered in Figure 2.6 based on the job numbering and not based on the priority in execution. This means that anchor three is going to be executed before anchor point one.

CONS performs very well. However, the restrictive approach that CONS applies, may lead to inefficiency in the system. Figure 2.9 presents such a case.

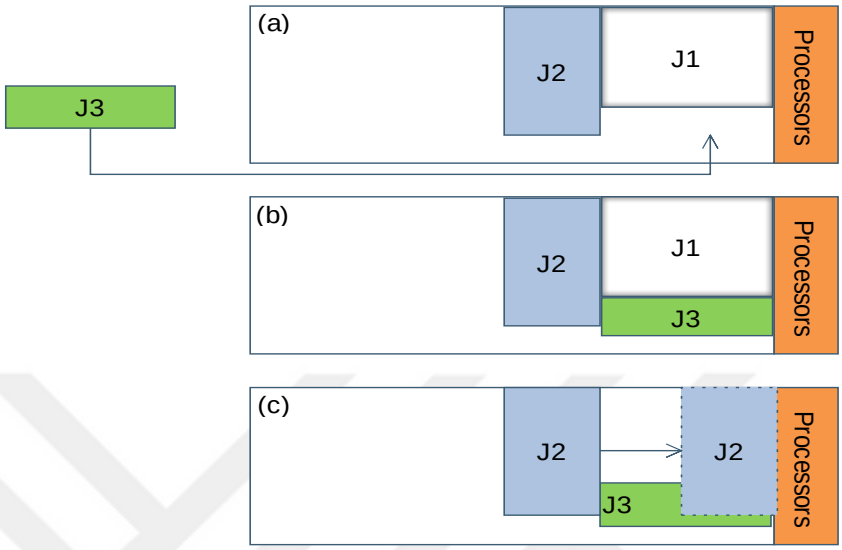


Figure 2.9. Pseudo delay

In the first case, case (a), J3 is selected to be backfilled since J3 will not delay J2. The backfilling will be executed as shown in case (b). J1 has finished earlier than expected and J2 cannot take the opportunity since J3 is being executed currently.

2.4.2 Extensible Argonne Scheduling System (EASY)

EASY [20] focusses on improving the utilization of the resources exponentially when the number of the jobs increases. EASY is an aggressive, robust and simple mechanism. EASY only concerns about the first job in the queue not to be delayed. While for the rest of the jobs, there is no guarantee about their starting time.

EASY categorizes the current running jobs based on their expected time to finish. Then, EASY searches for extra processors that can handle the first job in the queue. Once sufficient processors are located, EASY sits the “shadow time”. The shadow

time refers to the moment when enough processors are available to process the first job in the queue. In case extra free processors are detected, EASY maintains them for backfilling purpose. Then, EASY searches repeatedly within the queued jobs for the purpose of backfilling, provided that the backfilled job does not require more than the current free processors and backfilled job running time has to be over by the shadow time.

The issue with EASY however, is it delays the jobs ahead in the queue in order to backfill the short jobs. Consequently, EASY cannot assure about the jobs starting times. Eventually all jobs will be processed. However, jobs that are highly required to be processed may delay for non-specific time. Figure 2.10 presents the approach of EASY.

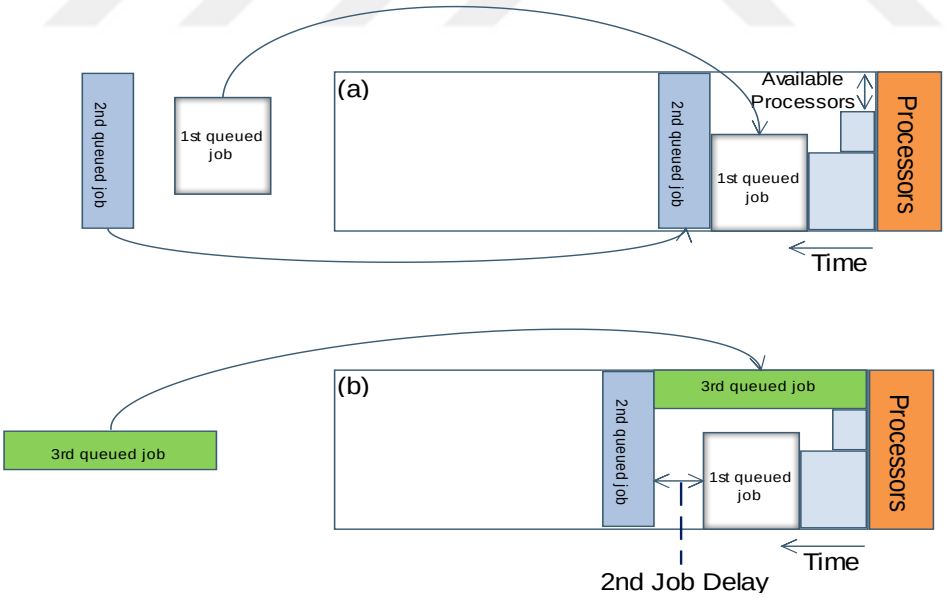


Figure 2.10. Backfilling in EASY

Figure 2.10 shows how EASY concerns only about the first job in the queue, whereas the rest of the jobs such as the second queued job; may be delayed if the backfilled job

requires more time to finish than the first queued job. EASY applies aggressive approach for backfilling purpose. However, this can be costly especially for the other jobs in the queue that belong to non-trivial application. Unknown delay time for the jobs strongly contradicts with the main essence of QoS, whereas in such concept, the users have to be aware when their jobs will start and the expected time to finish.

2.4.3 Swift Scheduler (SS)

Swift Scheduler mechanism (SS) [56] is a combination of SJF and heuristic search algorithm. This mechanism performs the scheduling for computational and memory resources. The main objective of this mechanism is to minimize the waiting time for the jobs in the queue. SS mechanism is modeled as follows: If the total number of jobs is $\mathrm{J_n} = [\mathrm{J_1}, \mathrm{J_2}, \mathrm{J_3}, \ldots., \mathrm{J_n}]$ and $\mathrm{R_m} = [\mathrm{R_1}, \mathrm{R_2}, \mathrm{R_3}, \ldots., \mathrm{R_m}]$, where *N* is the number of total jobs and *M* is the number of total resources, the overall completion time will be:

$$\sum_{i=0}^{N}\sum_{j=0}^{M} F\ (J_i, R_j) = \sum_{i=0}^{N}\sum_{j=0}^{M} G\ (J_i, R_j) + \sum_{i=0}^{N}\sum_{j=0}^{M} H\ (J_i, R_j) \tag{2.2}$$

where $\sum_{i=0}^{N}\sum_{j=0}^{M} G\ (J_i, R_j)$ is the expected completion time to finish using SJF mechanism and $\sum_{i=0}^{N}\sum_{j=0}^{M} H\ (J_i, R_j)$ is the heuristic function. The expected completion time is:

$$\sum_{i=0}^{N}\sum_{j=0}^{M} G\ (J_i, R_j) = \sum_{i=0}^{N}\sum_{j=0}^{M} \frac{J_{Li}}{R_{Cj}} \tag{2.3}$$

where $\mathrm{J_{Li}}$ is the job length and $\mathrm{R_{Cj}}$ the resource capacity. The heuristic function is:

$$\sum_{i=0}^{N}\sum_{j=0}^{M} H\ (J_i, R_j) = \sum_{i=0}^{N}\sum_{j=0}^{M} \frac{J_{Li}}{R_{Cj}} + Commov \tag{2.4}$$

where Commov is the communication overhead.

From Equations (2.3) and (2.4), the overall job completion time will be:

$$\sum_{i=0}^{N}\sum_{j=0}^{M} F\ (J_i, R_j) = \sum_{i=0}^{N}\sum_{j=0}^{M} \frac{J_{Li}}{R_{Cj}} + \sum_{i=0}^{N}\sum_{j=0}^{M} \frac{J_{Li}}{R_{Cj}} + Commov \qquad (2.5)$$

Equation (2.5) can be described as follows:

$$\sum_{i=0}^{N}\sum_{j=0}^{M} F\ (J_i, R_j) = 2\sum_{i=0}^{N}\sum_{j=0}^{M} \frac{J_{Li}}{R_{Cj}} + Commov \qquad (2.6)$$

In SS, the users send the jobs to queue and the jobs will be ordered based on SJF scheduling mechanism. The heuristic search selects the resources and then the scheduler maps the job that was scheduled using SJF with the chosen resource by the heuristic search. Figure 2.11 presents the principle of this mechanism.

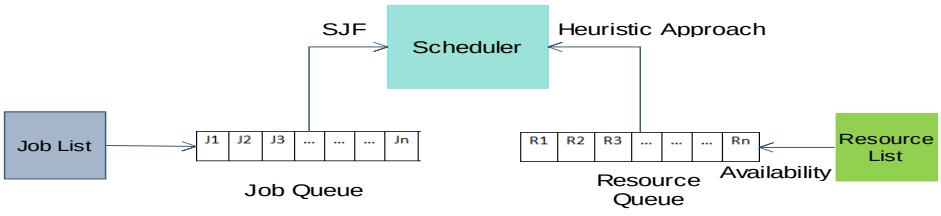


Figure 2.11. Swift Scheduler working principle

This mechanism outperformed the classical mechanisms, such as SJF, FCFS and Simple Fair Task Order Scheduling (SFTO). However, this mechanism suffers from several flaws. The utilization is low compared to any backfilling mechanism. In addition, the simulation did not cover real scenario, where there are thousands of jobs

and resources are there. The scenario was simple compared to the complexity of grid. Moreover, the authors did not clarify what is the specific heuristic approach they applied and what does the commination overhead in Equation (2.4) refers to.

2.4.4 The Biggest Hole (BH)

The Biggest Hole Priority Rule (BH) [19] is an extension of the Earliest Gap Earliest-Deadline First (EG-EDF) technique proposed in [11]. EG-EDF scans the schedule for the first gap that can fit the short jobs, whereas BH scans the gaps in the scheduler for the biggest hole that can handle short or even large jobs. The priority in BH technique is to detect the largest gap regardless the location of the gap. BH uses the backfilling technique and it aims to backfill any size of the job, unlike most backfilling mechanisms that tend to backfill only the short jobs. Figure 2.12 shows the difference between both mechanisms.

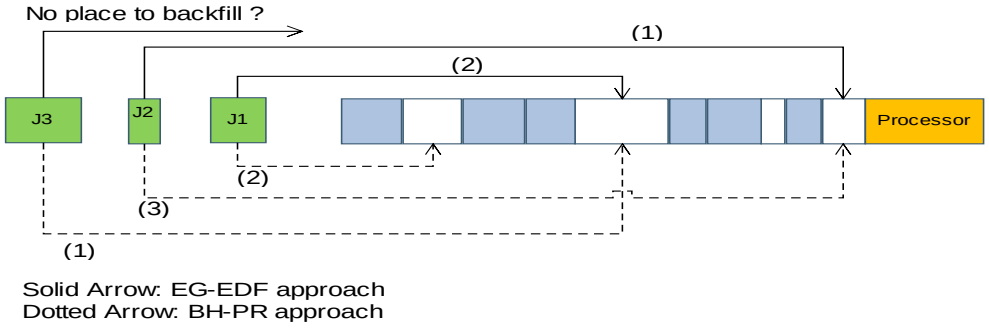


Figure 2.12. The difference between BH and EG-EDF in selecting the gap

BH mechanism was integrated with several priority algorithms such as SJF, Longest Job First (LJF), Earliest Release Date (ERD) and Minimum Time to Due Date (MTTD). The measured performance metrics are makespan, tardiness, response time, number of delayed jobs and utilization. In general, BH-SJF outperformed all the other integrated mechanisms including the original EG-EDF [11]. This mechanism performs

very well when the type of the jobs are fairly distributed in the system, i.e. when the workload contains short jobs and large jobs equally.

However, this mechanism gives the short jobs no priority since it backfills the jobs in the biggest gap and the biggest gap can fit both small and large job. In HPC environment, this will lead to reduction of efficiency of the system, since an overwhelming number of the jobs is short. Moreover, in real grid systems, performance metrics such as slowdown and waiting time are more important than makespan and BH-PR was not tested based on these aspects, which are very crucial in real grid environment. Finally, the maximum number of jobs that simulated in this work is 2500 jobs, which does not reflect the heavy load and the dynamicity in real grid computing environment [8].

2.4.5 Min-Min followed by Load Balancing

In [57], the authors have integrated Min-Min approach with Max-Min approach. Min-Min approach is well known as it causes high makespan, whereas Max-Min approach has high response time [58]. The main idea is to apply Min-Min in the first stage. In the second stage, the smallest jobs will be moved to the lightest resources.

First, the completion time for all jobs will be calculated. Then, the jobs will be mapped with the corresponding resources. The mechanism identifies the less completion time value and the most loaded resource. The most loaded resource can be identified by calculating the response time. The resource with the most response time value, is the most loaded one. Then, the proposed mechanism removes the minimum execution time from the maximally loaded resource. The following pseudo code described the proposed mechanism.

Algorithm 2.1 **Min-Min followed by Load Balancing**

1. for jobs [J_1,…J_Q] and resources [R_1,…R_N] do
2. calculate the expected completion time;
3. detect the resource that has less completion time for each job;
4. detect the job that has less completion;
5. allocate job to resource;
6. update the completion time value and resource capacity;
7. detect the heavily loaded resource R_{max} and the lightly loaded resource R_{light};
8. detect the minimum execution time loaded Job_{min};
9. move Job_{min} on resource R_{max} to R_{light};
10. end for

The proposed mechanism is implemented using GridSim Toolkit [59] in two scenarios. The first scenario is when the size of the jobs is small, whereas the second scenario contains large jobs size. In both scenarios, the proposed mechanism outperformed the original Min-Min and Max-Min for makespan, response time and improved the utilization of the total system. This mechanism is simple and efficient for small grid system.

The main drawback of this mechanism relates to the implemented scenario, which is limited and it does not simulate a real grid environment. The experiment only consisted of three resources and six jobs. Moreover, the mechanism was evaluated against Min-Min and Max-Min only. The authors claimed the proposed mechanism improves the QoS. However, enhancing QoS requires a simulation scenario, which is close to the real systems. In addition, other performance metrics have to be considered such as slowdown and tardiness [8].

2.4.6 Conservative Backfilling Extensions

CONS was extended in [60]. The first extension is Flexible Conservative backfilling (Flex-CONS), whereas the second extension is CONS followed by Gap Search (CONS-GS). Flex-CONS is a modification of the original CONS. In CONS, the jobs are backfilled without a priority, whereas in Flex-CONS the backfilled job are being backfilled based on a certain priority.

The jobs are scheduled based on the scheduler's goal and they are prioritized in the schedule based on their deadlines, waiting time and the execution time. The possibility for the job to get the chance to be backfilled depends on how close is the job's deadline, how long the job spent to be scheduled and how short it is.

Hence, the job that has close deadline or spent long time in the schedule and its execution time is short, has the priority to be backfilled. Flex-CONS has extra feature compared to the original CONS as described before. However, since Flex-CONS is just a modification of CONS, it inherits the flaws of CONS. Figure 2.13 illustrates the concept of this mechanism.

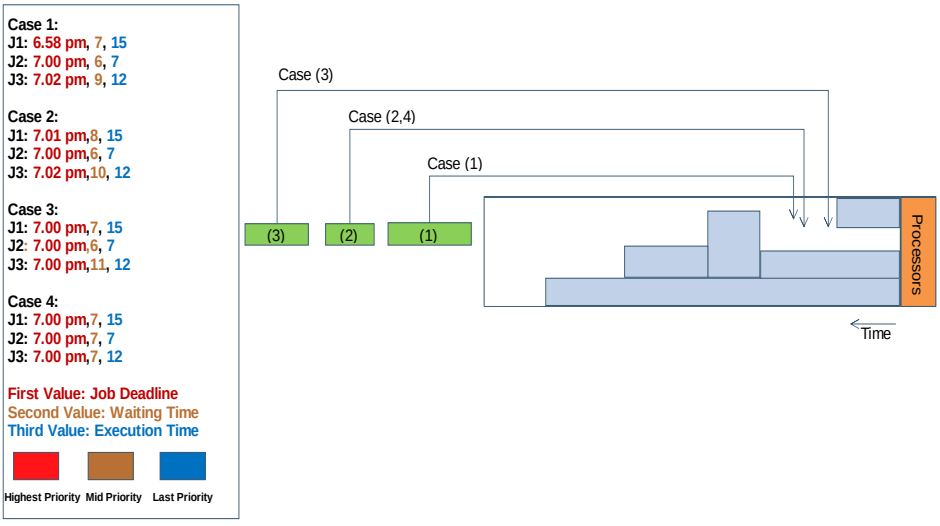


Figure 2.13. Backfilling decision in Flex-CONS

CONS was integrated with Gap Search (CONS-GS) in [61]. Gap Search optimizes the solution that was obtained from CONS. The optimization criteria in Gap Search are waiting time, response time and slowdown. CONS-GS performs very well in real grid environment. This mechanism outperforms CONS, EASY and Flex-CONS. However, due to the restrictive nature of CONS, the optimized solution that obtained from CONS can be further improved.

2.4.7 Related Works Summary

Table 2.2 summarizes all the reviewed mechanisms features, types, strengths and weaknesses. The table summary shows that CONS-GS is the best mechanism among the reviewed ones, since it uses backfilling and optimization technique as well as it is implemented using planning scheduling approach.

Table 2.2
Mechanisms properties

Mechanism	Scheduling Approach	Backfilling Aspect	Heuristic/ Optimization	Strengths	Weaknesses
CONS	Planning	Yes	No	No delay for the jobs ahead	Restrictive scheduling
EASY	Queues	Yes	No	High Utilization	Backfill with delay
SS	Queues	No	Yes	Simple	Low Utilization
BH-PR	Planning	Yes	No	Suitable for short and long jobs	Not suitable for HPC systems
MIN-MIN+LB	Queues	No	Yes	Simple and efficient for small system	The considered criteria doesn't simulate real grid system
Flex-CONS	Planning	Yes	No	Backfilling with priority together	Restrictive scheduling

Table 2.2 continued.

CONS-GS	Planning	Yes	Yes	Solution optimized	First solution is based on restrictive approach

2.5 Tabu Search (TS) for Optimization Purpose in Scheduling

TS [62, 63] is meta-heuristic mechanism that performs local search to dig for new solutions. TS escapes from optimal local solution, where the mechanism keeps providing the same solution. In such a case, the mechanism is trapped in a loop and no further improved solution can be found [64]. TS avoids this trap thanks to the adaptive memory, which helps in creating more flexible search behaviour. TS has the ability to explore new solutions for applications that are based on linear, nonlinear and stochastic functions.

The following sections briefly reviews this mechanism. First, the main components of this mechanism is presented, followed by search strategy. Finally, an example about how this mechanism reduces the penalty is presented.

2.5.1 Tabu Search Components

TS is based on three main components, which are the adaptive memory, an associated mechanism, and integration in the memory functions. The adaptive memory built on structure enables the historical search and evaluation criteria information to be exploited totally, unlike other memory structures, which are inflexible, or other systems that does not have a memory.

The associated mechanism is mandatory in order to implement the memory and to control the interaction between the constrains and the loose search procedure. The

integration of memory functions enables TS to apply short memory term or long memory term in different periods of time. The ability to apply short and long memory allows TS to apply intensive and diverse search. Intensive search supports the move combinations, whereas diverse search lead the mechanism to explore new regions.

The short memory leads the mechanism to search in aggressive way in order to get the best moves. The best move subjects to the desired criteria where these criteria existed in the restrictions in Tabu. The restrictions are intended to guarantee that searching direction will not be repeated in some particular moves. This is achieved by banning the attributes of these moves to be executed.

The main role of the restrictions in Tabu is to go beyond local optimum solution to perform high quality moves. Without the restrictions, Tabu may take the best move away from the local optimum, by applying a move without any improvement. Thus, Tabu will take the best move in that point and consider it as best move, so that the mechanism falls in the local optimal trap. Certainly, these restrictions are designed in a manner of application required criteria. Figure 2.14 shows the short memory component.

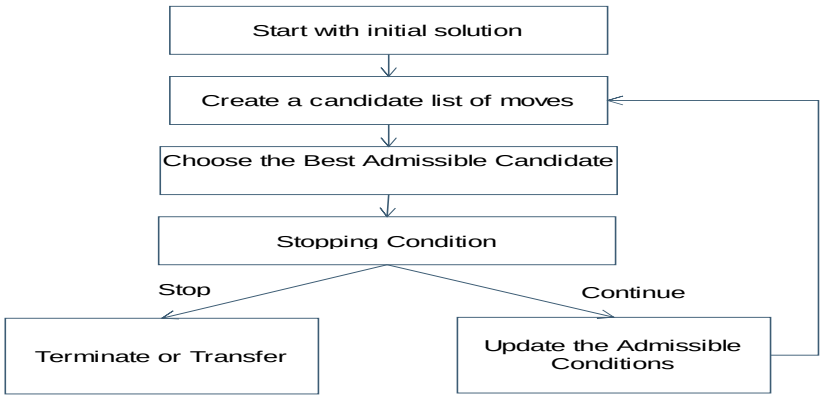


Figure 2.14. Tabu Search short memory component [62]

From Figure 2.14, the working steps for the short memory term can be summarized as follows. The start is based on the current solution; the current solution refers to the obtained initial solution. The initial solution was generated from a previous mechanism or long-term memory component. In the second phase, every new move represents a new solution based on the initial solution. The selection of the new solution will be based on Tabu restrictions. The Tabu restrictions are based on the required criteria demanded from the application. The Tabu restrictions will be updated once a new better solution is found. The newly obtained solution will be saved if it was better than the previous ones. The search will stop based on the stopping conditions. If the number of requested iteration is achieved, then the search will stop or if the optimization time is over, then the best solution will be selected. Alternatively, the solution can be transferred to the long-term memory for more intensive and diverse search [62].

2.5.2 Selecting the Best Candidate Move

This step is very crucial as the evaluation of the move will be based on the criteria required. In every move, the value of the required criteria that are related to the objective functions will be measured. In addition, based on the evaluation value, the move will be categorized as acceptable move or not, i.e. the search will determine each calculated value before and after the evaluation. In some cases, when going for another move is difficult because it is hard to determine which move has to be executed, an approximate solution can be used for evaluation purpose.

The approximation is a method used when solving the problem becomes hard. Thus it solves a nearby problem, which is easier to solve. The information obtained from solving the nearby problem will be used to solve the original problem. However, as long as the search for new moves is going on, the search becomes more adaptive. This

is due to the number of moves that are listed in Tabu list are relatively fewer than the number of available move. Figure 2.15 illustrates the selection on the best move.

From Figure 2.15, the steps for finding the best move can be summarized as follows. In the first stage, an evaluation process starts for the current solution. If the current solution is not the best one and there is no better solution (move) left, the current move will be selected as best move, otherwise the move will be examined if it belongs to Tabu list or not. If the move belongs to Tabu, the solution will be tested if it satisfies the criteria or not. If the move does not belong to the Tabu list, it will be accepted immediately. In selecting the best move process, there is a loop that always keeps checking if there is no better move left. Once there is no better move left, the loop ends and the best move will be selected.

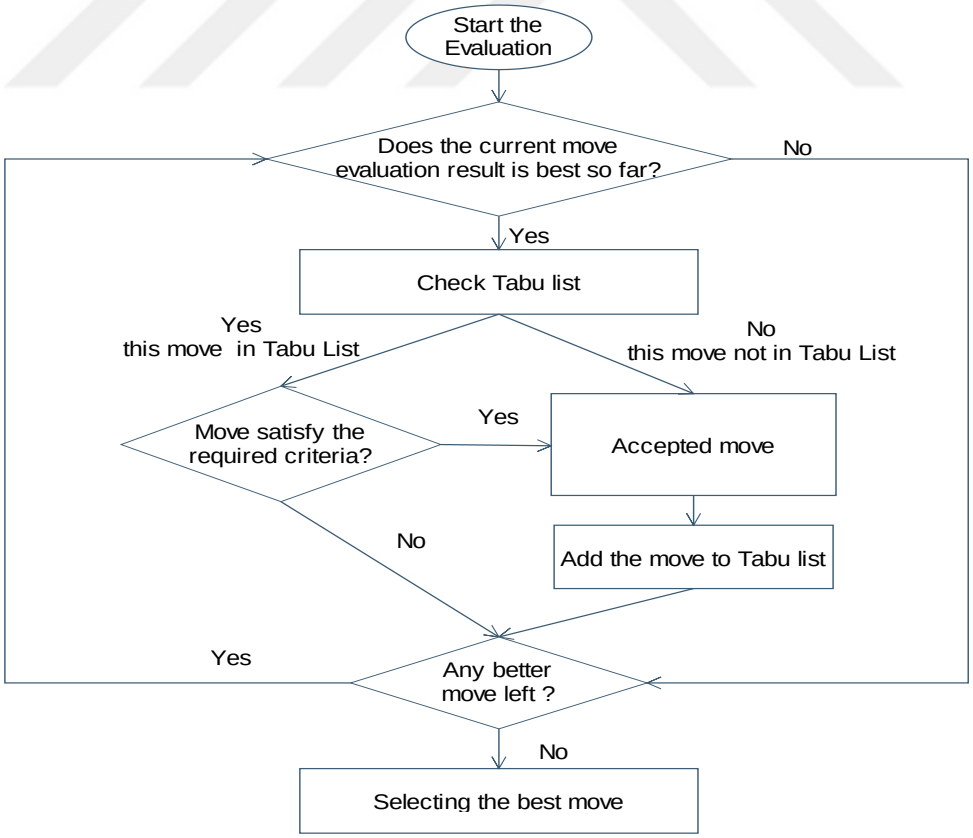


Figure 2.15. Selecting the best proposed move [65]

2.5.3 The Iterations in Tabu Search

This simple example illustrates how TS tries to find a new solution in each iteration. Figure 2.16 [66] presents the iterations and the Tabu solution in each iteration. The following hypotheses have to be set before explaining each iteration. This example is applied in minimum cost spanning tree. The minimum cost spanning tree means that all the nodes have to be connected with each other without any loop in a minimum cost. There are five nodes and two constrains. The first constrain is at most one of the three edges $X1, X2$ and $X6$ can exist, which means two or more edges will violate the constrain. The second constrain is if edge $X1$ exists, that means edge $X3$ has to exist either. The violation cost of each constrain is equal to 50. The constrains can be represented by the following expressions:

$$\begin{cases} X1 + X2 + X6 \leq 1 \\ X1 \leq X3 \end{cases} \tag{2.7}$$

The straight line in Figure 2.16 refers to existence of the edge in the tree where its value is equal to one. The dotted line refers to the absence edge in the tree, where its value is equal to zero. The solution is applied based on “swap edges”, i.e. adding one edge will lead to deletion of another edge. Tabu move is valid for two iterations only and in the third iteration it has to be removed. Tabu move can be removed in the second iteration only when a better solution is found. The less cost path refers to the best solution. Finally, Table 2.3 summarizes the hypotheses of the explained example.

Table 2.3
The hypothesis of iteration example

Hypotheses	Hypotheses Values and Info
Environment of example	Minimum cost spanning tree
Cost	X_1: 6, X_2: 9, X_3: 18, X_4: 2, X_5: 1, X_6:8, X_7: 12
Number of constrains	2

Table 2.3 continued.

Definitions of constrains	- At most, X_1, X_2 and X_6 can exist at the same time. - If X_1 exists, X_3 must be existed in the tree
Penalty of each constrain	50
Repeating Tabu solution Limitation	2
Solid line	Existing edge
Dotted line	Dropped edge
Solid line value	1
Dotted line value	0
Solution technique	Swapping edges

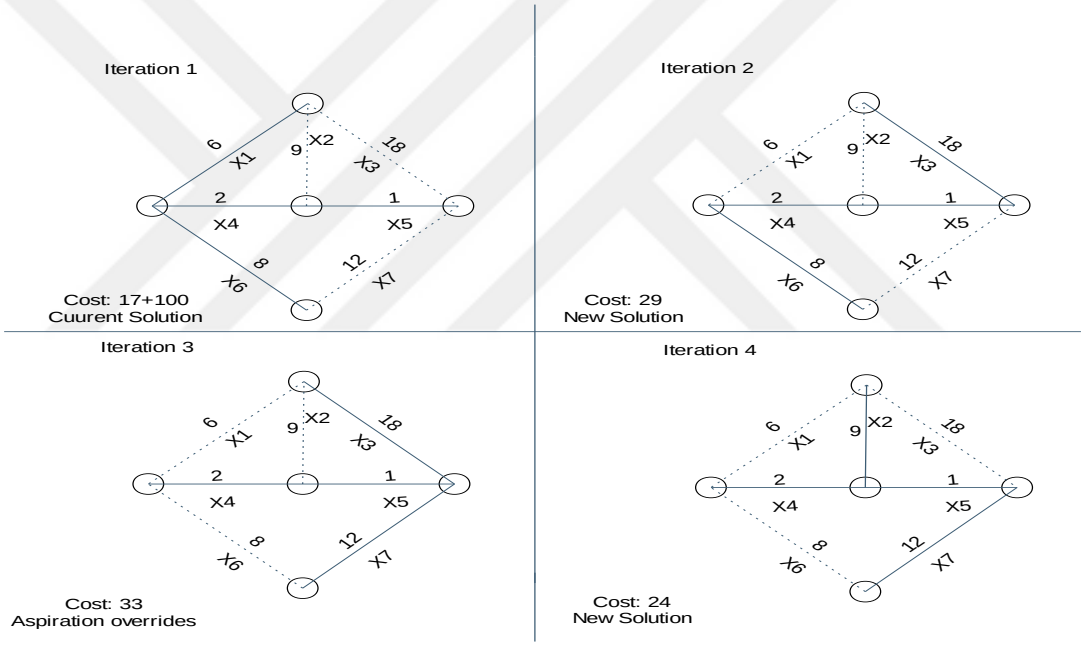


Figure 2.16. Iterations in Tabu Search

• **Iteration One:** In the current solution, the cost is equal to 17. However, the value 100 is included to the 17 due to violation on the two constrains. X1 and X6 exist in the tree together and X1 exists in the tree, while X3 is not. Adding X3 and deleting X1 will increase the cost to 29, but it will avoid the penalty.

• **Iteration Two:** In this iteration, X3 is a “Tabu solution”, since it was included to the tree in the previous iteration. To maintain the “Tabu Solution”, all the nodes will be connected together without a loop and to avoid the constrains, X7 will be added to the tree and X6 will be dropped. The total cost will then be 33.

• **Iteration Three:** X3 is a “Tabu solution” and hence it cannot be dropped in this iteration only if dropping it provides a better solution, does not violate the constrains and connect the five nodes without a loop. Thus, X3 will be dropped and X2 will be included instead. The total cost is 24, which is the current best solution.

Iteration Four: Both edge X7 and X2 are now “Tabu Solution”. The best solution is to drop X2 and add X3, but this cannot be done since X2 belongs to “Tabu Solution”. Moreover doing this step is going back to iteration three. Thus, the best solution will be deleting X5 and adding X3. This is the only solution in order not to violate the constrains while maintaining the “Tabu Solution” and keep the whole nodes connected without a loop. The total cost is equal to 42.

Tube Search is being implemented widely in several fields and different scopes, such as scheduling, routing, graph optimization, power distribution and telecommunication [64]. TS was evaluated for scheduling purpose and it was found that this mechanism significantly enhances the scheduling quality by its unique approach and its diverse search in order to come up with a new solution [67, 68].

2.6 Summary

This chapter presents a comprehensive review related to the issues that are covered in this study. First of all, a background about grid computing is provided followed by the working steps of grid technology. The concept of jobs allocation was shown with a

general clarification related to the quality of allocating the jobs to the resources. Then, a comprehensive review related to scheduling approach is presented in order to justify the implementation of the planning approach. The related work covered well-known and recent mechanisms, which are related to the scope of this study. Each mechanism was addressed by providing its approach followed by the strengths and weaknesses. Finally, a brief review about TS is provided, by highlighting the general approach of this mechanism in order to provide good reasons for applying it in this study.

CHAPTER THREE
RESEARCH METHODOLOGY

The main objective of this study is to improve the Quality of Service (QoS) for the end users in grid computing environment for High Performance Computing (HPC) applications. The improvement in QoS for the end user can be achieved by minimizing the delay in the scheduler. The objective of this chapter is discusses the methodology used in carrying out the research, whereby its purpose is to present the approaches used in executing the research and to demonstrate how the approaches can deliver to meet the objectives of the research. A robust and solid methodology that fits this study is required and hence this study implements Design Research Methodology (DRM) [69] by applying its steps along with the introduced work stages.

This chapter is organized as follows; it points out the research approach in Section 3.1. Section 3.2 presents the initial stage of DRM named Research Clarification (RC). Section 3.3 highlights the second stage named Descriptive Study-I (DS-I). In this stage, the general research steps followed by conceptual model are addressed. Section 3.4 describes the methods adopted in designing the proposed Swift Gap (SG) and Completion Time Scheme (CTS) mechanisms in line with the third stage of DRM, named Prescriptive Study (PS). Section 3.5 presents the describes the simulated system, the workload properties and clarifies the experiments' settings alongside with evaluation performance metrics. Finally, this chapter is concluded in Section 3.6.

3.1 Research Approach

The main objective of this study is to propose a mechanism that improves the QoS for the end user in grid computing environment for HPC applications. This requires adequate awareness and well correspondence between existing mechanisms and the

developed mechanism to get better results [70]. Applying multi-level scheduling by providing initial solution, and improving the obtained initial solution using the optimization technique, would be the approach to achieve this objective. The optimization technique was implemented in several studies that tackled the scheduling problem in grid computing [71-73]. Moreover, Completion Time Scheme (CTS) will be applied to further improve the performance metrics that are related to the QoS.

To do so, the research has to be designed accordingly. Blessing and Chakrabarti [69] have included all the needs and requirements for the design research. According to Blessing who defined the design research: "design research integrates the development of understanding". These features are orchestrating with each other to assist in producing an efficient solution, which would result in an improved performance mechanism. Blessing assured that design research should be scientific to obtain suitable outcomes in both theoretical and practical science [69]. Therefore, a particular methodology is required.

Consequently, Blessing came up with DRM. DRM is an approach, guideline and a set of methods that can be implemented as a framework for making design research. DRM allows the researchers to demonstrate the design research more carefully and accurately. DRM proved its efficiency and success [69]. Due to of these attractive features, DRM would be adopted for conducting this research. The objectives of DRM are mentioned as follows:

a. To produce a framework for researchers to conduct design research.

b. To assist in catching the research areas that is academically and practically valuable and practical.

c. To enable several of research methods in addition to help in selecting proper methods and groups of methods.

d. To produce a strategy for precise research and methodical planning.

DRM is split into four different stages which are RC, DS-I, PS and DS-II. Figure 3.1 clarifies the DRM framework where it displays the links among DRM stages besides the basic methods implemented in each stage, and the major outcomes. The thin arrows among the stages clarify the main process outflow, whereas the thick arrow to and from each stage explains the methods that were used and the outcomes of that precise stage.

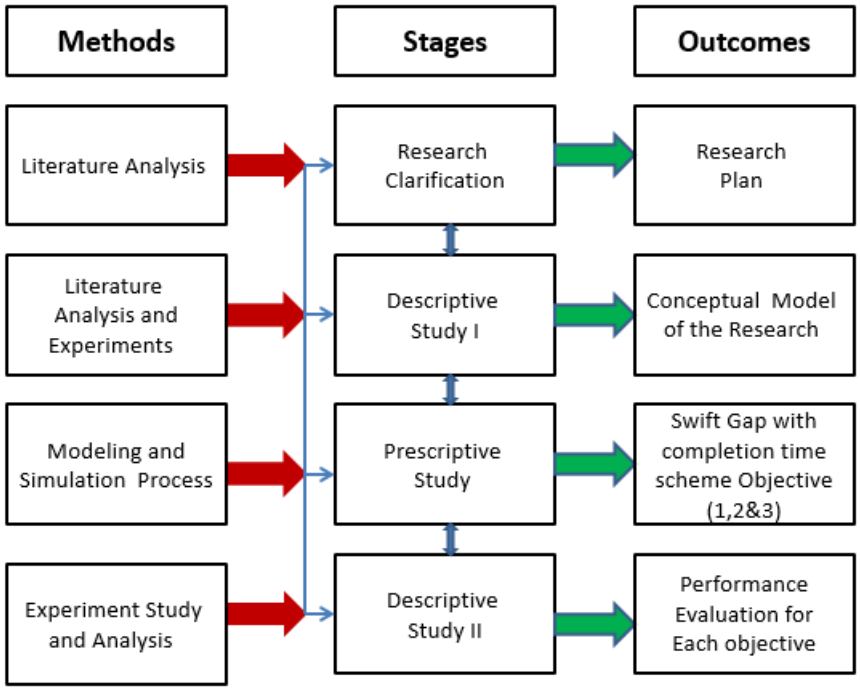


Figure 3.1. Research approach [74]

3.2 Research Clarification (RC)

RC is the prime step of DRM. This is the starting point of the research. RC allows researchers to identify and understand the problem to be tackled prior to creating a research plan. Figure 3.2 explains this stage.

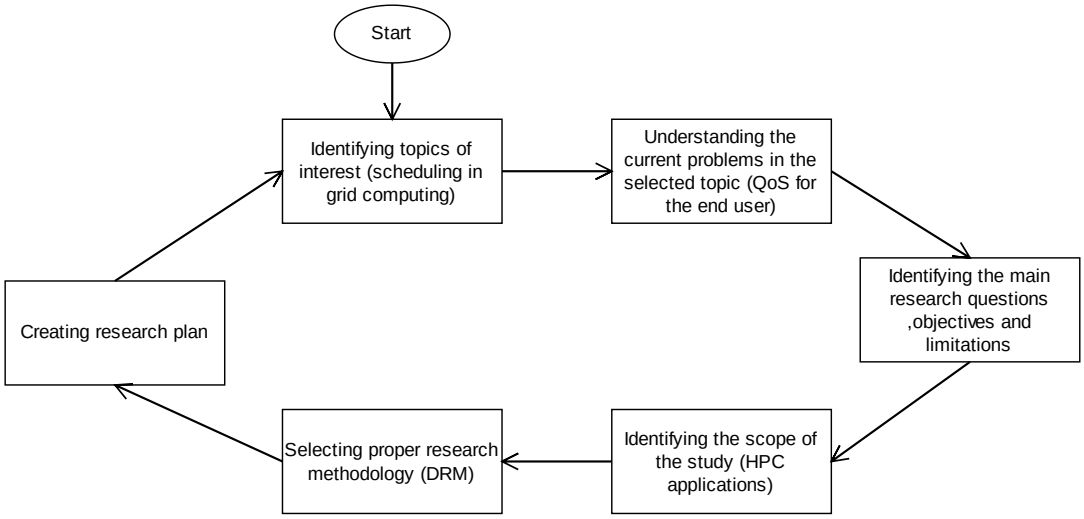


Figure 3.2. Crucial steps in research clarification

RC involves six refined steps, as shown in Figure 3.2. Usually, the main outcome of the RC stage is Chapter One. Particularly, the outcome of this stage are the followings:

a. The research topic and motivation.

b. Current issues, statement of the problem, research questions and objectives.

c. Scope and limitation of the research.

d. Significance, contributions and achievements of the conducted research.

3.3 Descriptive Study-I (DS-I)

After the accomplishment of the RC stage which created the whole research plan, the research shifts to the second stage, which is known as DS-I. DS-I contributes in understanding the current situation deeper. In addition, DS-I engages in a critical review of the literature that are related to the research area and experimental studies. In this research, a comprehensive review of the current proposed mechanisms is discussed and several experimental studies are critically evaluated [75] to ensure better

and deeper understanding of the current solutions and novel trends. Figure 3.3 presents the five iterative steps in DS-I. Each step attempts to increase the level of understanding and potentially lead to further experimental studies or literature reviews, which would improve and update the performance and conceptual models.

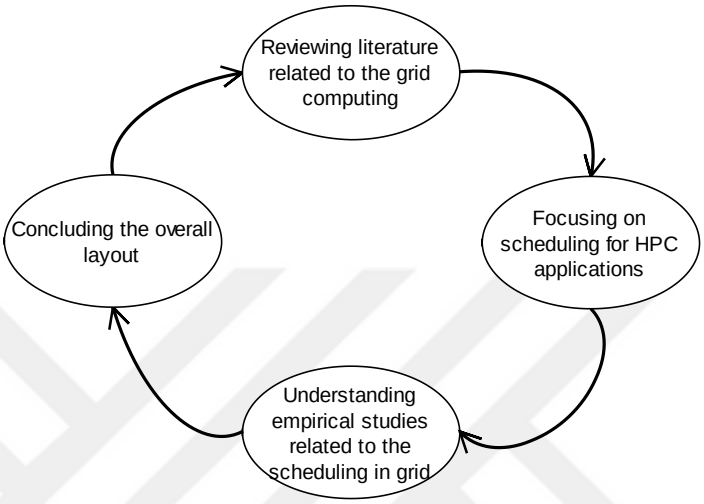


Figure 3.3. Main steps in DS-I

The outcomes of DS-I stage are:

a. Critical review of previous work related to resource allocation mechanisms as presented in Chapter Two.
b. SG conceptual model.

3.3.1 Conceptual Model

The conceptual model of SG comprises of two parallel columns. The first column is the mechanism itself, while the other column is the CTS. SG is an integration between backfilling mechanism and optimization mechanism, whereas the CTS covers two parameters, which are the start time and the processing time. Figure 3.4 shows the conceptual model of the introduced mechanism.

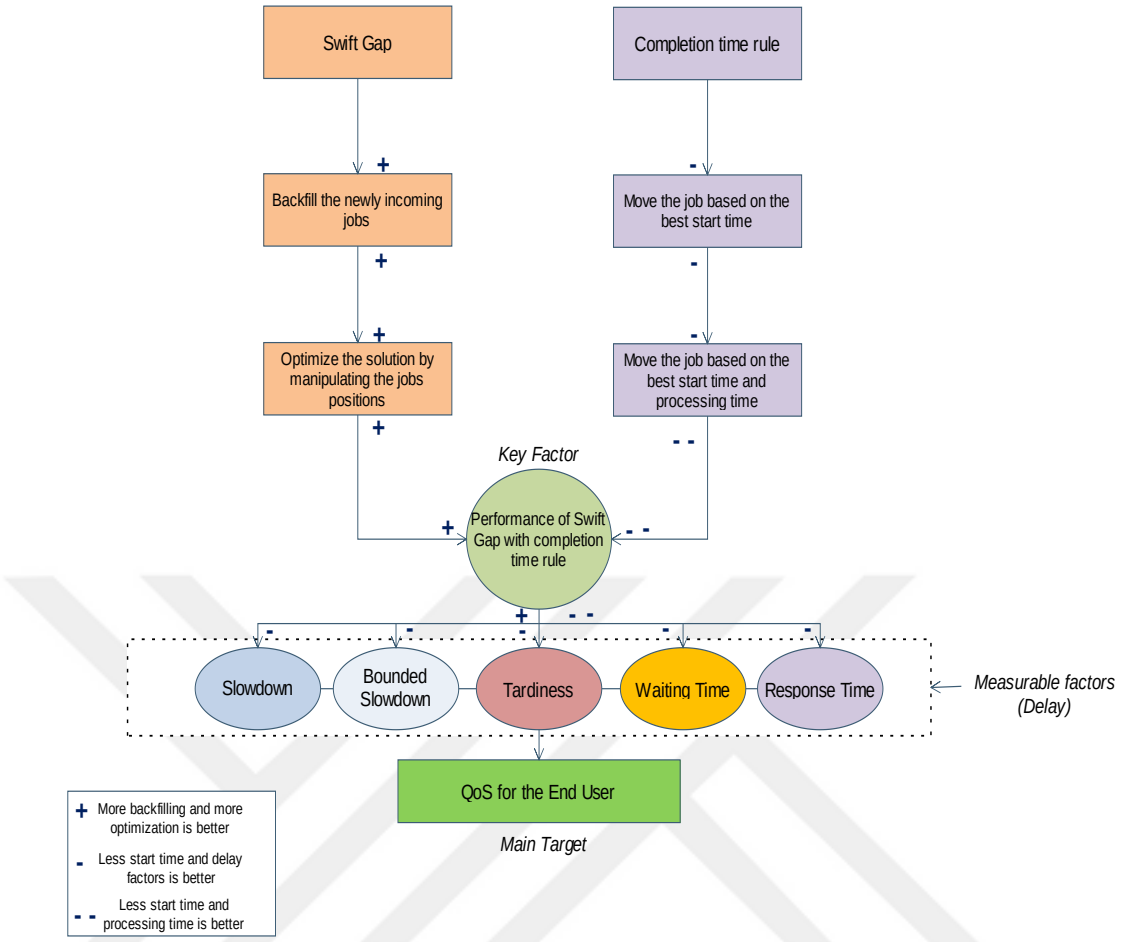


Figure 3.4. Conceptual model of full SG mechanism

As shown in Figure 3.4, it can be observed that the more the number of backfilling and optimized jobs, the better the result will be. This is due to positive effect of backfilling and optimization on the measurable factors, which have to be reduced. While in the CTS, the less the time spent for the jobs to be completed results in less delay for the jobs. Hence, the positive sign (+) is used for backfilling and optimization, i.e.; the more the better, whereas negative sign (-) is used for CTS, i.e.; lesser time is better. Moreover, since the delay factors are presenting the time perspective for the jobs, therefore less time that represented as negative (-) sign, is considered as measurable success factors. Finally, the main target in this study is to achieve high QoS level for the end user, hence, the QoS is considered as the main success factor.

3.4 Perspective Study (PS)

This stage is crucial and important in DRM, as it incorporates the design of the introduced SG mechanism. To achieve this purpose, the hybridization scheme and simulation process have to be conducted [8, 76]. The first block describes the features of the proposed mechanism SG. The second block shows modelling development, which involves problem analysis, goals aim and study of related theory. The third block presents the integration of the introduced mechanism (SG).

Furthermore, this block describes the hybridization with details in Chapter Four. The fifth block (5a and 5b) presents the CTS and the parameters included in the developed scheme. The sixth block explains how this scheme is applied and linked with SG mechanism. The last block is related to the verification and validation of introduced work. Figure 3.5 presents the PS stage.

The outcomes of the PS stage are:

a. The implementation of SG mechanism and CTS.
b. Chapter Four (objective one), Chapter Five (objective two) and Chapter Six (objective three).
c. Verification and validation of the introduced work.

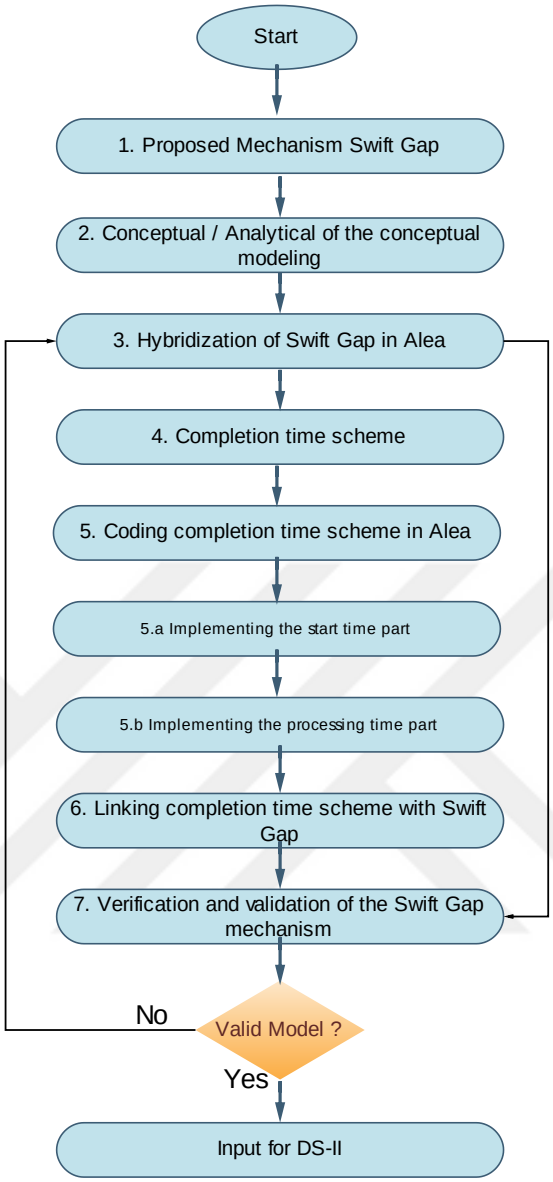


Figure 3.5. Perspective study main stages

3.4.1 Verification and Validation

Model verification evaluates the integrity of the transformed model that was illustrated through flowchart or pseudo code to an executable computer program [77]. The simulator used in this research was built based on Java. Therefore, in this research, the structure of the proposed mechanism will be implemented using Java as a programming language. In addition, all mechanisms must be verified to assure that

code was written correctly without errors or bugs [78]. NetBeans Integrated Development Environment (IDE) [79] is implemented for this purpose. Many functionalities are provided by the NetBeans IDE for Java developers such as, editing, debugging, launching, parsing as well as generating for creating files. NetBeans IDE supports Java, PHP, C, C++ and HTML5.

In order to verify SG code with CTS, QJ-Pro software is used [80]. QJ-Pro is a code analyser for Java-based platform. QJ-PRO analyses the code construction, displays the errors and provides solution for them. This software analyser checks the code for potential bugs, misapplying Java language and the coherence of coding standards. QJ-Pro is installed as an extension to Eclipse Java Development Tools (JDT) [81]. Figure 3.6 confirms that SG and CTS are implemented correctly and the coding structure is free from errors and bugs.

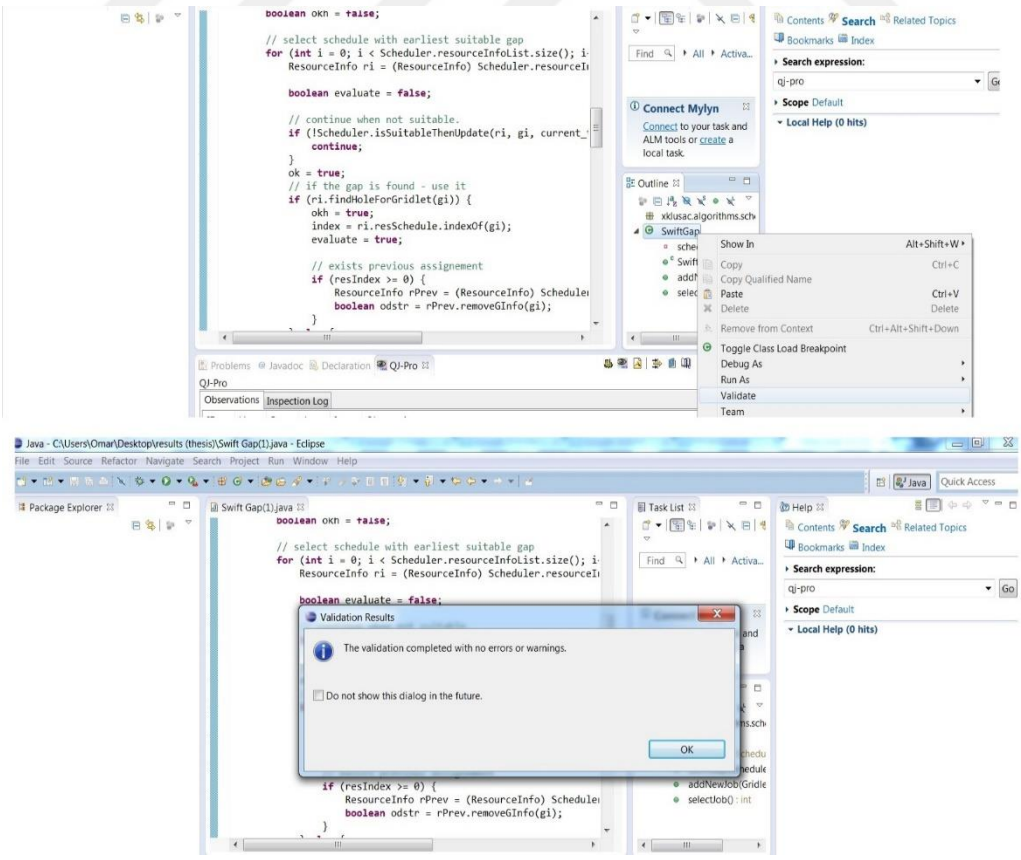


Figure 3.6. Code analysis in QJ-Pro

Balci in [77] defined validation as the affirmation of the implemented model, policy or a mechanism that acts accurately compared to other validated models. The accurate ratio has to be acceptable and the behaviour has to be consistent with models and simulations. Hence, the validation perspective is about to build a right model. The right model means the introduced mechanism performs the basics and required functions properly.

Validation techniques were discussed in [77, 82]. These methods are informal, formal, static and dynamic. Each main method has sub-techniques. The informal method depends on the human factor; neither rigorous mathematical rules nor guidelines exist in this method. Informal method is usually applied in robust approaches based on formal guidelines. Audit technique [83] is one example of informal method.

The formal method is based on mathematical proof. If the mathematical model of proof is correct, that means the model is valid. However, not all models are obtainable to prove mathematically their correctness. In addition, based on the current state of the art, many formal techniques are not applicable for complex and reasonable simulation models. Inductive and inductive assertions are examples of formal method techniques [83, 84].

The static method concerns about the truthfulness of model's aspect. This method does not need an actual implementation of the model. Rational implementation is sufficient to validate the model. The static mode has information about the flow structure of the model, the source code and data. This method can be implemented for automated tools. The simulation compiler represents [83] and Cause-Effect Graphing are examples of this method [85, 86].

Dynamic method requires an implementation for the model. The validation is based on the behaviour of the implemented model. The dynamic method is implemented through three phases. The first phase is adding a source code into the body of the executable model. The inserted codes collect information about the implemented model. Then, the model is executed. Finally, the output of the executed model will be compared with other valid models. The verdict of the validation will be based on the behaviour of the output model compared to the valid models [77, 87]. Alpha testing [88] technique is an example of this method.

Since this study is focused on scheduling the jobs in dynamic environment, the dynamic method will be implemented to validate the SG mechanism. The integrated mechanism output will be compared with the output of other validated mechanism using the same simulation tool based on the same experiment setup and environment. The validation will be implemented by simulating the mechanism for numerous times using different numbers of jobs to measure the sensitivity of the mechanism regarding changing the number of the simulated jobs [89, 90].

The results that are generated from simulating SG will be presented graphically. Then the behaviour of the SG will be compared with valid mechanisms [85, 91] based on the graphical lines [92-95]. The validation of SG touches three techniques in dynamic approach, which are Sensitivity Analysis [89, 90], Graphical Comparisons [92-95] and Comparison Testing [85, 91]. The CTS that is developed to further improve the performance is implanted based on the weight function expressions [96-99]. Since the weight function is a widely used and well known function, hence no further validation is required.

3.5 Descriptive Study-II (DS-II)

The Descriptive Study-II (DS-II) focusses on the evaluation of the designed mechanisms and application. Performance evaluation is a very serious step for evaluating any types of research. The evaluation of application performance may be carried out using three possible traditional methods [100-102], which are simulation, analytical modelling and measurement.

3.5.1 Evaluation Approach Consideration

Choosing the evaluation technique is an extremely important step in all performance evaluation projects [103]. Table 3.1 shows the strengths and weaknesses of performance evaluation techniques as stated by Jain and Raj in [103]. From Table 3.1, and according to Buyya et al. in [59, 104, 105], it is quite clear that the simulation would be the most suitable approach for the performance evaluation of this research.

Table 3.1
Comparison of performance evaluation techniques

Criteria	Analytical modeling	Simulation	Measurement
Time Required	Low	Medium	High
Accuracy	Low	Moderate	High
Tools	Analysts	Computer Languages	Instrumentation
Trade-off Evaluation	Easy	Moderate	Difficult
Cost	Small	Medium	High

3.5.1.1 Analytical Modelling

Analytical or mathematical modelling is a set of mathematical equations that describe the performance of the system based on its actual case [54]. Through computer programming, a mathematical model can be investigated. To achieve that, computer

programming translates the operations by using functional relationships within the system. The results of the mathematical model can be represented by a graphical form, which was drawn from the output of the running program. This approach is very efficient when studying the system behaviour of the very rare or unsafe in real life.

This technique is often used to study simple systems where an analytical model will be built and validated to explore and solve a specific problem in a system. That would help for better understanding the conceptual view of the system before moving to the implementation part. Hence, this approach is efficient for simple systems. When the complexity of the system increases, this technique would seek for simplification and assumptions to concentrate on specific aspects of the system and fix the rest. According to Jain in [103], mathematical modelling has plenty of benefits and advantages such as low cost, less time required and easy in trade-off evaluation. However, analytical modelling has low accuracy as compared to other techniques in performance evaluation.

3.5.1.2 Measurement

Measurement based performance evaluation can be carried out using test-bed or implementation of a real network. In addition, this approach offers very accurate results but the expenses of this approach are high due to the specialized tool's needed [106]. Moreover, structuring a real network test bed for a certain scenario is usually expensive and has limitations in terms of working scenarios, mobility models and distributed systems and so on. Furthermore, the measurements in this approach are usually not repeatable.

3.5.1.3 Simulation

Simulation approach is commonly used for representing dynamic responses and behaviours for real systems. It is used as a model from computer-based system, or generated through computer programming. Moreover, to study the performance of mechanisms and protocols, simulation is considered as a flexible tool to investigate about performance. It allows analysis of the performance in scalable and repeatable environments [107, 108]. Thus, this research will use simulation to measure and evaluate the performance of the computational grid.

Here are some of the advantages of using a network simulator to study computational grid performance.

a. Simulators require a single computer to execute the simulation experiments and analyse the acquired results.

b. Simulators enable network researchers to apply various scenarios in a short period of time.

c. Complicated topologies can be easily built via simulation environment, whereas such topologies would be hard to replicate in a real environment.

d. Simulators offer access to data about all traffic that were transmitted during the simulation experiment.

3.5.2 Evaluation Environment

In a large-scale distributed computing environment such as "Grid", simulation is the preferable approach in order to test and analyse the performance of the scheduling

algorithms. Moreover, simulation can bypass the challenges that related to highly parameterized problems. In this type of environments, the number of jobs and resources is massive and the environment is complex. Therefore, in order to obtain better and expressive results, the experiments must be repeated several times. In addition, it is extremely hard to get a smooth access to available and ready grid platforms to evaluate the algorithms due to both resources and user's tasks keep changing profusely [109]. Thus, the simulation approach could be the most widespread method in order to conduct these types of studies. Table 3.2 shows the comparison among the several simulators that will be mentioned later.

Table 3.2
Simulators specifications

Simulator	Aim	Language	Limitation and Features						
			X1	X2	X3	X4	X5	X6	X7
GridSim	Scheduling in computing grid	Java	✓	✓	×	×	×	×	×
Alea simulator	Scheduling in computing grid	Java	✓	×	✓	×	×	×	✓
Sim-G-Batch grid simulator	Simulating energy and security	C++	✓	✓	✓	✓	✓	×	×
Balls simulator	Peer-to-peer structure	Java	✓	✓	✓	×	×	✓	×

X1: Centralization scheduling supporting, X2: Decentralization scheduling supporting, X3: Dynamic scheduling supporting, X4: Security supporting, X5: Energy awareness supporting, X6: Load Balance supporting and X7: Complex simulation using real workloads in HPC application in grid computing environment.

There are numerous discrete event distributed computing simulators available for computational grid environment [59, 110-115]. Each simulator has its own unique advantages and disadvantages like data replication, workload trace-back and differentiated scheduling QoS in order to obtain an overview about the scheduling quality for both admin and user.

Several studies were conducted to compare the performance of these simulators. Several grid simulators are built based on each other; for instance: Alea simulator [116] is based on GridSim [59], MaGate Simulator [117] is based on the GridSim as well and Sim-G-Batch Grid simulator [118] is based on HyperSim-G [119].

3.5.2.1 Alea Simulator

Alea simulator [116] is based on GridSim 5 Toolkit simulator [59]. Alea is centralized grid scheduler that implements advanced and complex scheduling techniques. One of the main features of Alea is its ability to deal with dynamic situation if additional jobs are suddenly involved in the system during the simulation. In such case, the generated schedule updates the job list over the time and thus, it can handle newly arrived jobs while the execution of old jobs is already done.

Alea simulator consists of eight main entities, which are the Scheduler, Grid Resources, Machine Loader, Failure Loader, Complex Gridlet, Result Collector, Job Loader and Visualizator. The Scheduler itself consists of two parts; the first part is the Communication, while the second part is the Scheduling. The Communication part is responsible for linking the Grid Resources with Complex Gridlet (the representative tasks) and submitting the results, which are received from the scheduling part to the Result Collector.

The Scheduling part is responsible for scheduling the jobs. Scheduling often depends on the strategy that used in the scheduler. The Grid Resource entity is in charge of simulating the Grid Resources, whereas Machine Loader creates the resources in the grid. Failure Loader simulates the case that if a failure occurred in the machine. Job Loader generates dynamic jobs from the workload trace while Complex Gridlet represents these jobs to the Scheduler. Once the simulation is done, the Result Collector saves the data for Visualizator and generates the final simulation results. Finally, Visualizator draws the graphs based on the obtained results [116].

By means of object-oriented paradigm that is utilized in the development stage, Alea can be smoothly extended. This is due to the functions are split in different classes and thus, any modification can be conducted smoothly without any conflicts with any other function that implemented in different class. Moreover, the scheduler itself is divided into two main parts as aforementioned, which are the Communication part and the Scheduling part. Therefore, extending or modifying the scheduling algorithm can be achieved smoothly and efficiently. In addition, if a new job type or a certain property is demanded, only the Gridlet class has to be updated [116].

Alea simulator was evaluated and received a positive feedback from many researchers around the world. The evaluation covered the performance of the simulator and the correctness of the generated results. The performance evaluation was conducted through two stages. The first stage involves complex real-life data, whereas the second stage tests the speed and the scalability of the simulator compared to GridSim based GSSIM and the GridSim toolkit.

The correctness of the obtained results was invariably validated. This is achieved by analysing the implementations and the output of the algorithms. The pseudo codes of

the scheduling mechanisms were checked and compared with their original pseudo codes [14, 120, 121]. The obtained results from Alea, were compared to the output of the original mechanisms for several trace files. When abnormal behaviour in the mechanism performance was detected, the implementation was traced and fixed accordingly using classical debugging techniques. Figure 3.7 shows the components of Alea simulator [116].

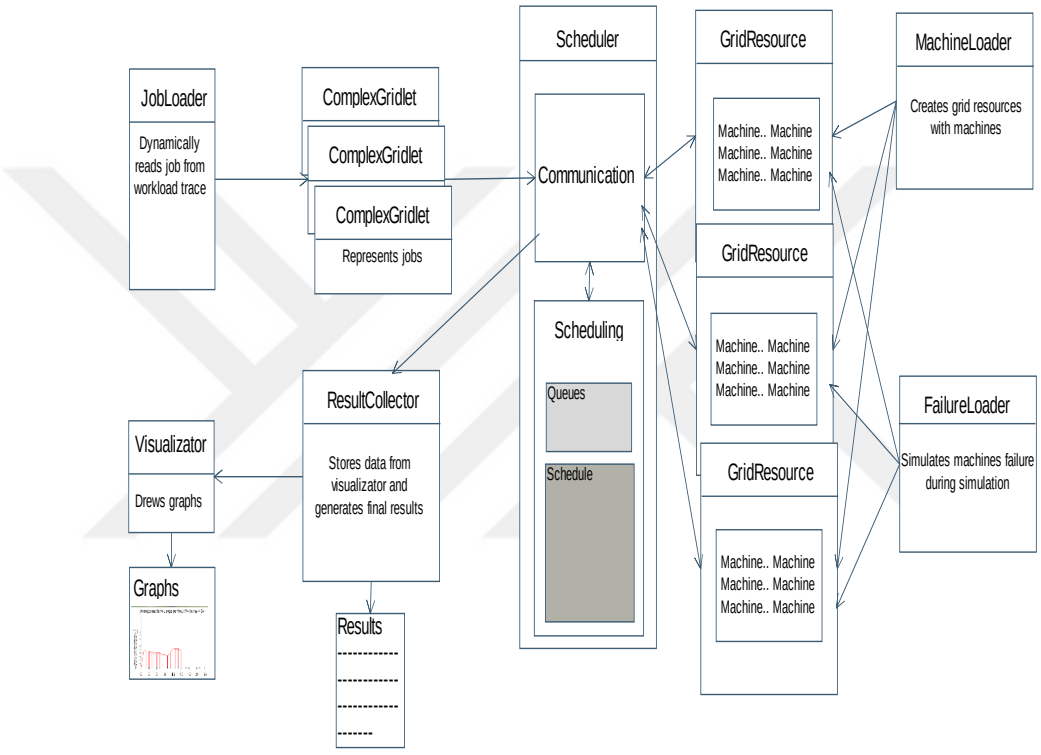


Figure 3.7. Alea simulator components [122]

3.5.2.2 Experiment Steps

Hassan and Jain in [123] has split the simulation stages into eight steps. Generally, the simulation task consists of two main stages, which are pre-software stage and software stage. Figure 3.8 presents these steps.

- Pre-software stage includes the following steps:
  a) Describe the study objective specifically.

b) Select fixed parameters, such as iteration number and simulation time.

c) Select performance metrics to evaluate the introduced mechanism SG. The selection of the performance metrics are based on real required criteria to fulfilment the QoS for the end user.

d) Select variable parameters, the objective of the performance evaluation is to study the influence of specific variables on the chosen performance metrics. The variable parameters could be the number of the jobs, different workload and different mechanism for evaluation purpose.

- Software stage includes the following steps:

a) Configure/program simulation to generate relevant performance metrics selected in pre-software stage step (c).

b) Run the simulation software (Alea). Once the simulation is over successfully, the performance metrics data are ready to be collected.

c) Present the data collected in the previous step in a meaningful format then explain the presented results.

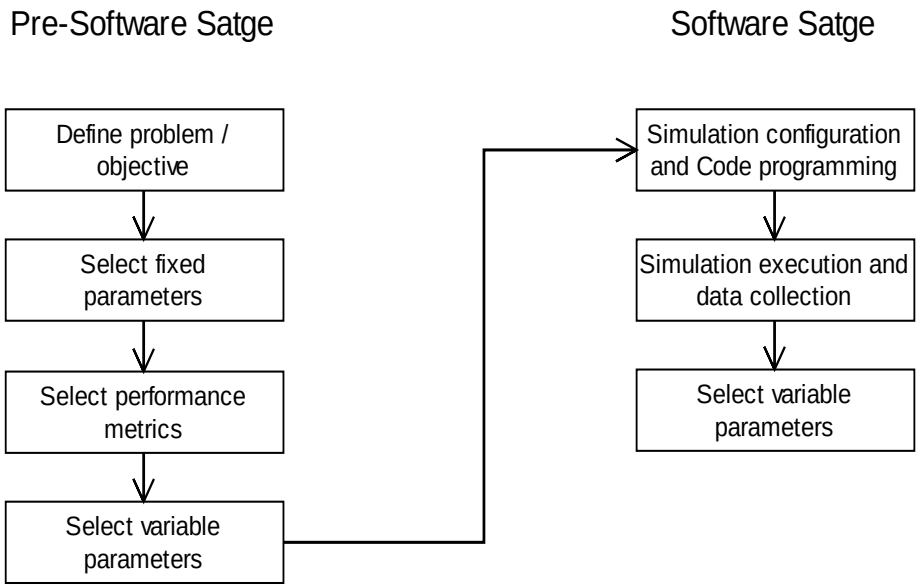


Figure 3.8. Simulation setup

3.5.2.3 Experiments Setup

In this study, all the experiments are conducted using Alea simulator [116]. Three workloads are considered to be simulated. Each workload represents an application. These workloads were traced and captured from real jobs in real grid system environment [124]. Since this study is implemented for HPC applications, hence all workloads traces that simulated in this research actually belong to HPC application. The jobs are coming to the system dynamically and the scheduler that applies the scheduling mechanism is centralized.

3.5.2.4 System and Workload Description and Experiments Settings

In this research, a classical scenario is implemented (see Figure 3.10), with one centralized scheduler that manages both jobs and the system resources. The jobs sent by the user may require one or more CPUs. Some parameters, such as job arrival time, job processing time and resources capabilities, are specified in the runtime estimation. There are no precedence constraints among the jobs.

Each cluster consists of several machines and each machine consists of several CPUs. The computational power within each cluster is the same. The space slicing processor policy [46] is enabled within the same cluster. This enables parallel execution if the requested amount of CPUs is lower or equal to the number of CPUs in the cluster. Thus, the machines can be co-allocated in the same cluster to handle a parallel job, although this cannot be executed if the machine belongs to a different cluster.

The experiments in this study are done with three workloads: Zewura, Wagap and Meta [125], containing 17,257, 17,900 and 495,000 jobs respectively [125]. Zewura consists of seven clusters, (Zewura1, ..., Zewura7). Each cluster has one machine

and 80 CPUs. Each cluster has 529,426,432 GB of RAM. The total number of CPUs is thus 560 and the total RAM is 3,705,985,024 GB [125].

Wagap has two clusters, Zewura and Zegox. The Zewura cluster consists of 20 machines each with 80 CPUs, totalling 1,600 CPUs; the total RAM 529,426,432 GB. The Zegox cluster consists of 48 machines each with 12 CPUs, totalling 576 CPUs; the total RAM in this cluster is 94,035,968 GB. Wagap has a total of 2,176 CPUs and the total RAM is 623,462,400 GB [125].

Finally, the Meta workload consists of 13 computational clusters. The first, Manwe, has seven nodes each with 16 CPUs, totalling 112. The Hildor cluster has 26 machine, Mandos cluster 14 machine, and. Haldir and Nympha 28 each. The scenario of the conducted simulation is addressed in Figure 3.9. The simulated workloads belong to HPC applications. The selection of these workloads is based on various cluster numbers in each workload, difference in jobs length and number of processing units required for each job. The rest of the specifications of this and the previous workloads are presented in Table 3.3.

Table 3.3
Workload properties

Workload properties	**Zewura**	**Wagap**	**Meta**
Number of clusters	7	2	13
Number of machines per cluster	1,1,1,1,1,1,1	20,48	7,26,14,49,28,19, 28,11,1,1,10,40,16
Number of CPUs per cluster	80	80, 12	112,416,896,588 224,152,112,88 32,64,160,320,192
RAM per cluster (TB)	529426432	529426, 94035	66060, 67108, 264241, ,19922, 14680,3395,14680,1058013,1040187,67108, 25165 ,50331

Table 3.3 continued.

Total CPUs number in the workload	560	2176	3356
Total RAM in the workload (TB)	3705985	623462	2690895
Speeds	1,1,1,2,1,1,2	1,3	1,1,1,2,1,4, 1,3,1,3,1,2,2
Number of recorded jobs	17.2K	17.9K	495.3K
Application Type	HPC	HPC	HPC

All the experiments in this study were conducted using Alea simulator version 3.1 on Windows 7, with an Intel I7-4770 CPU with 8 GB of RAM. The speed of job arrival times to the system is fixed at 1.0, the simulation time is 3,000 seconds and the optimization time limit is 200 seconds. The optimization was iterated 100 times. The substantial factor of user estimation time is two. The bounded slowdown threshold (ξ) is 10 [126]. The simulated resources are CPUs. Both parallel and sequential jobs were simulated. The settings of all the parameters are presented in Table 3.4.

Table 3.4
Experiments setup settings

Simulation Settings	**Value**
Simulator	Alea 3.1
Simulated Jobs	3000-350000
Simulation Time	3000 Sec
Optimization Time Limit	200 Sec
Simulated Resources	CPU
Runtime Estimates	Enabled
User Runtime Estimates Factor	2
Clusters Speed	Enabled

Table 3.4 continued.

Due Date (di)	2*Runtime
Simulated Workloads	Zewura, Wagap, Meta
Jobs Arrival Time	1.0
ξ	10 Sec
Iterations	100
Experimental Device	Intel Core I7-4770 @ 3.4 GHZ, 8 GB RAM, Windows 7

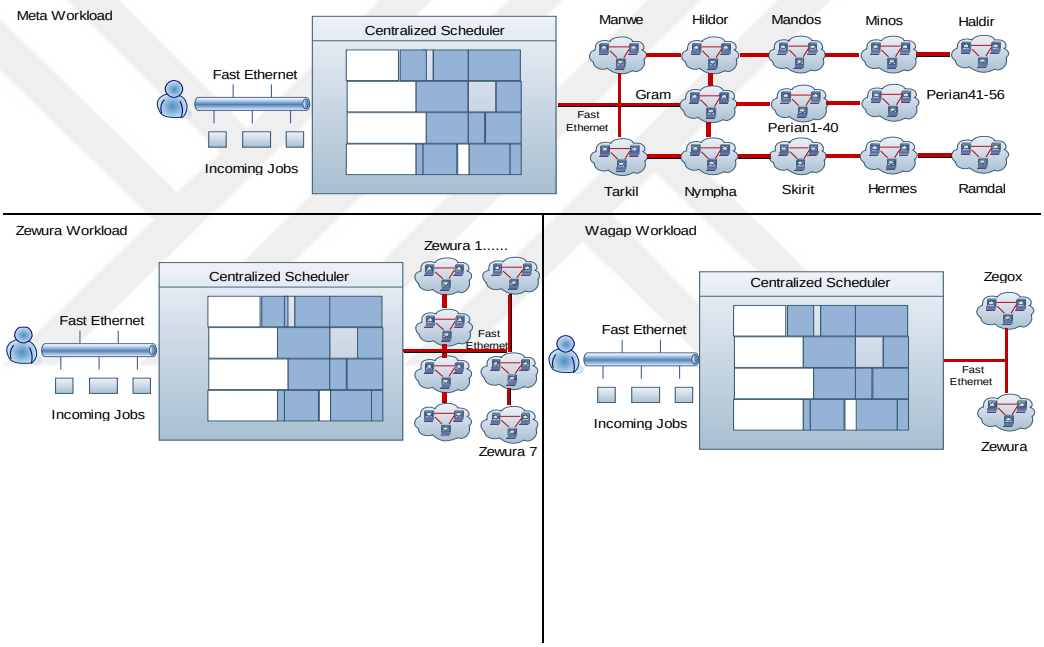


Figure 3.9. Simulation scenario

3.5.2.5 Performance Metrics

Performance metrics may refer to other signification in different domains based on the context in which they are used. For this research, the performance metrics that considered are Slowdown, Bounded Slowdown, Tardiness, Waiting Time and Response Time. The selection of these metrics is based on the studied literature, whereas numerous studies recommended considering these performance metrics for

real grid environment that runs for long time, especially from the perspective of the end user [8, 15, 126].

- Slowdown (sld): This metric refers to how many times the job was delayed. This metric has no unit. The numerator is normalized by the denominator and both units are measured in seconds. The formula [127] as shown below.

$$sld = \sum_{j=1}^{n} \frac{Completion\ Time - Submission\ Time}{Completion\ Time - Start\ Time} \tag{3.1}$$

  The expression of slowdown can be also given in the following formula.

$$sld = \sum_{j=1}^{n} \frac{Tr + Tw}{Tr} \tag{3.2}$$

where Tr is the running time of the job and Tw is the waiting time.

- Bounded Slowdown ($bsld$): The slowdown metric is affected sharply of the very short jobs. For example, if there is a very short job with a running time equal to 10 milliseconds (ms) and the waiting time is equal to 10 minutes, which means the slowdown for this job is equal to 6000. Whereas, for jobs that have longer execution time (10 seconds for example) and the waiting time is equal to 10 minutes, the slowdown will be equal to 60. Although the slowdown measures all the jobs, however it can be affected by very short jobs as discussed above.

  The bounded slowdown [46] reduces this effect by dividing the numerator over "fixed threshold" rather than actual time for the very short jobs. From Equation (3.3), it can be observed that the value of the "fixed threshold" crucially affects the behavior of this metric. The "fixed threshold" value is recommended to be equal to 10 [126].

$$bsld = \sum_{j=1}^{n} max\left[\left(\frac{Tr + Tw}{\max(\Sigma, Tr)}\right), 1\right] \quad (3.3)$$

- Tardiness (tar): This metric refers to the delay of the completion time of the jobs for the corresponding due date. The formula for tardiness is given [128, 129].

$$tar = \sum_{j=1}^{n} max\,(0, Cj - di) \quad (3.4)$$

where Cj is the completion time for the job and di is the corresponding due date.

- Waiting Time (Tw): This matric shows the time that jobs spent waiting in the system, from the submission moment until the processing starting time. The waiting time can be calculated using this expression [128, 129].

$$Tw = Start\ Time - Submission\ Time \quad (3.5)$$

- Response Time ($Respt$): This metric expresses about the total time that the job spent from the moment it was submitted to the system until its execution time is over. This metric is represented by adding the execution time to the waiting time [126].

$$Respt = Tr + Tw \quad (3.6)$$

where Tr is the running time of the job and Tw is the waiting time.

3.6 Summary

This chapter has described in detailed information related to the methodology that was implemented in order to conduct this research. DRM methodology is applied in this

research due to its robustness, flexibility as well as plenty of previous empirical studies have implemented this methodology [74]. DRM consists of four main stages. The first stage is represented in the plan of this study. The outcome of this stage is the statement of problem, which pointed out the current issues related to the scheduling from the perspective of the end user in dynamic grid computing environment in order to achieve high QoS level. The second stage presents how the conceptual model of the proposed SG algorithm is built based on the current issues, the understanding of the related work and the proposed solution to improve the performance.

Perspective Study stage is crucial, as this stage shows how SG is going to be implemented and how the CTS is going to be developed. Moreover, the verification and validation steps are explained in order to show the dignity of the introduced work. The final stage is Descriptive Study II, whereas the evaluation of SG and CTS is described. Since the simulation evaluation approach has low cost in addition to the repeatable feature compared to other evaluation approaches, therefore the simulation is implemented in this research to evaluate the outcome of this research using real workloads.

The following chapters explain in great details the designing process of SG mechanism and the development of the CTS.

CHAPTER FOUR
SWIFT GAP MECHANISM

This chapter explains the Swift Gap (SG) mechanism, which uses the backfilling technique to schedule jobs. SG implements the schedule-based system to gain the advantage of setting all the jobs in advance, using runtime estimates. An optimization is applied to further enhance the scheduling. This chapter details the design of the proposed mechanism, and is organized as follows. Section 4.1 presents the schedule-based approach in grid computing. The description and design of the SG mechanism are introduced in Sections 4.2 and 4.3. Verification and validation of SG are presented in Sections 4.4 and 4.5 respectively. The evaluation of SG is provided in Section 4.6. The summary of this chapter is presented Section 4.7.

4.1 Analysis of Approach

In queue-based systems, when jobs arrive in the grid system, the scheduling decision is taken immediately. When the number of jobs or the resource reservation increases, the queue-based system suffers from lack of Quality of Service (QoS) for non-trivial applications [50].

Queue-based systems have no information about the start time for the job, the finishing time or when the job will be processed. If high-priority jobs reach the system, i.e. jobs that have a deadline, an advance reservation of the resources is needed. However, in this case queue-based systems can not perform well, due to lacking of information about the parameters required to reserve the resources in advance. Obviously, in such cases, poor performance, such as high response time and slowdown, affects the QoS [130].

In the schedule-based approach, all parameters are known in advance, and thus the scheduling decision is set in advance; the known parameters are number of available resources, the computational power of the resources, the number of jobs and the execution time for each job. For non-trivial applications, the schedule-based approach is efficient since it enables planning for future scheduling and therefore works very well for the grid environment, especially when high QoS is required. Figure 4.1 shows the schedule-based approach compared to the queue-based system.

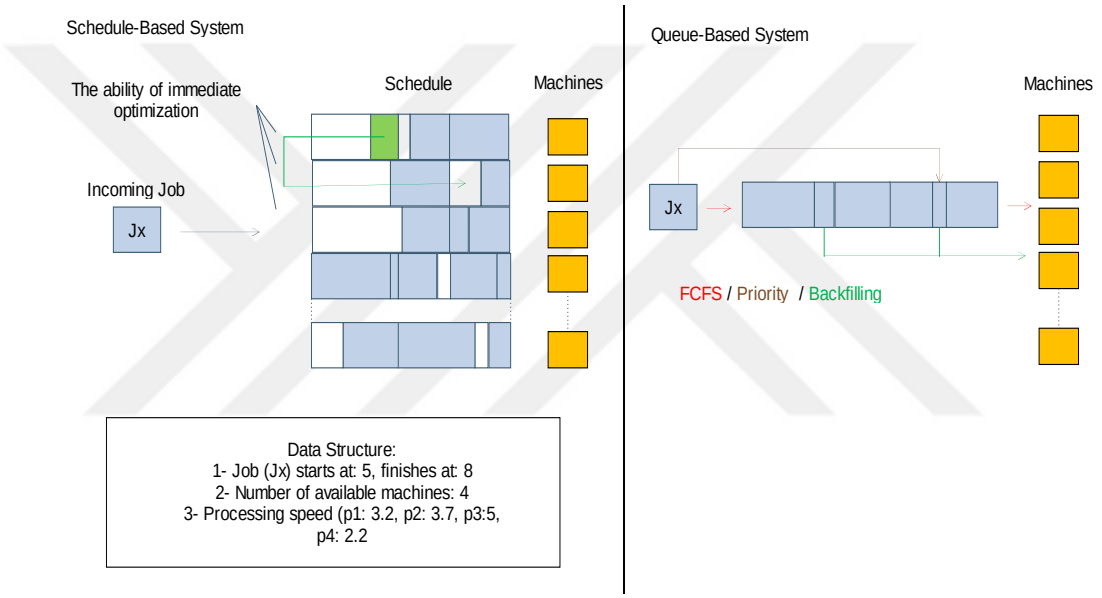


Figure 4.1. Scheduling system vs Queue system

Since the scheduling parameters are provided as mentioned before, this approach enables the system to predict the next system state. For instance, during peak hours, the system can limit requests within a certain range using “dependent limits”. Dependent limits are a set of rules controlled by the administrator, based on certain agreements between the service provider and the end user. Thus, the administrator can limit a specific user or a particular project to control machine usage during prime time. Thus, the schedule-based approach improves the load balance of the system during

peak hours.

Figure 4.2 shows the dependent limit concept controlled by the administrator in the schedule-based approach for two projects, A and B. For instance, project A can use 60% of available resources in the worst case and up to 75% in the best case. Project B has fewer dedicated resources (25% in the worst case and up to 50% in the best).

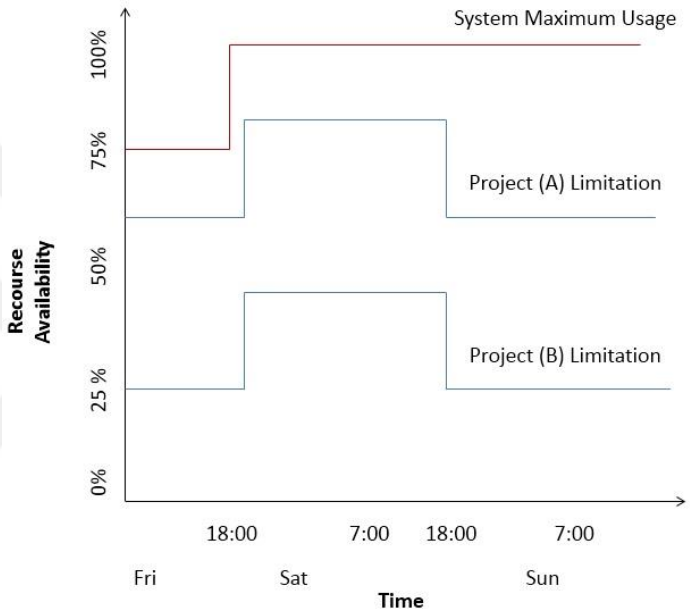


Figure 4.2. Resource management

Even though the schedule-based approach has many advantages, it was originally designed for static systems. In a fully dynamic environment such as the grid, many urgent events related to resources or jobs may occur. The machine can fail, restart itself, or there may be an error in job arrivals. In addition, high-priority jobs may reach the system all at the same time. In these situations, when a sudden modification takes place, the schedule which was set in advance may fail to match the new updates in the system. In order to resolve this issue, the schedule structure was improved to meet the system updates using two techniques, the incremental schedule and the anytime

approach [131].

Several common events may be implemented by the scheduler, corresponding to the dynamic grid behaviour. These events are job arrival time, job finishing time, resource failure and resource restart. If a new job arrives in the system, there is no need to compute the schedule again. Re-computing the schedule is costly and is at the expense of computational processing time. When the incremental schedule is applied then a new state of the system is detected; the incremental schedule just has to update the existing schedule about the new event. This means that the incremental schedule reuses the current schedule rather than computing a new one from scratch every time a job reaches the system. This technique ensures the minimum number of steps required to keep the system updated [131].

Further optimization improves the initial schedule and fixes some issues related to resource failure and resource reboot. In each iteration, local optimization checks whether any jobs are affected by the resource failure, whether a new resource is available in the system, and whether that new resource has a vacancy to tackle a new job. Applying the incremental schedule alongside the optimization results in quick local changes caused by the dynamic behaviour of the grid. Thus, the schedule will be up to date with a minimum number of computational steps required.

Some alterations, such as job arrival error, resource failure or restart, and high-priorityjobs that reach the system unexpectedly, are handled by adopting the anytime approach. This can be used even if optimization is currently applied. However, unexpected events may suddenly occur, and the anytime approach enables the mechanism to stop at any optimization stage immediately and flexibly.

This technique is very efficient in a fully dynamic grid environment, since many changes can occur, or an event may suddenly arrive. Therefore, in order to be aware of this change or urgent event, the optimization can be stopped at any time. Thus, and if necessary, a swift decision can be taken to handle the new event and keep the system always updated regarding any modification. Figure 4.3 illustrates the interactions among the three discussed approaches.

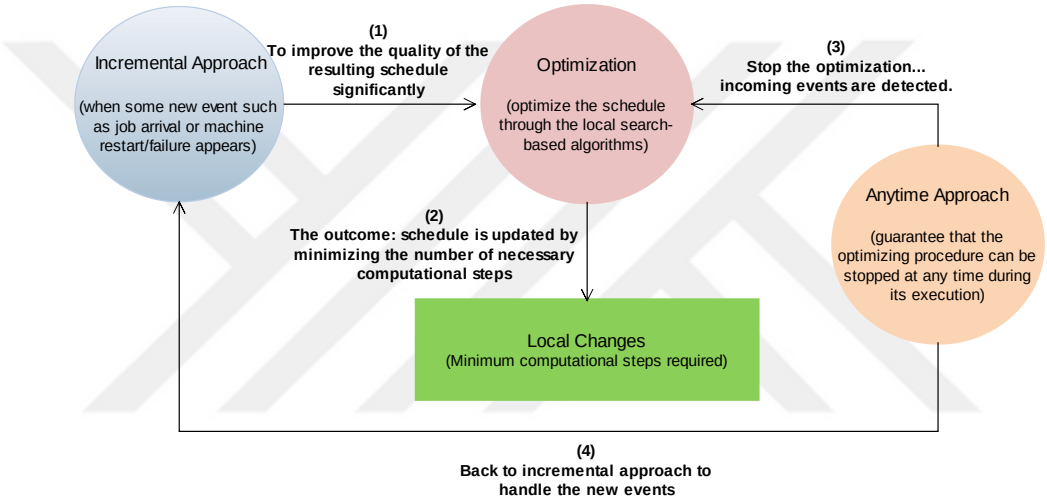


Figure 4.3. Updating the scheduler

4.2 Swift Gap Mechanism

SG mechanism implements the schedule-based approach in backfilling fragmentations (gaps) in the schedule by moving the short jobs ahead without causing any delay to the large-sized jobs. This differs from other existing mechanisms, such as the Extensible Argonne Scheduling System (EASY), which backfills the gaps by moving forward the jobs arbitrarily. EASY is only concerned that the first job in the queue is not delayed, while the rest of the jobs will be delayed, causing a long waiting time for all the jobs in the queue, including short jobs, with a high average slowdown.

SG avoids this by enabling the runtime estimation provided by the schedule-based approach. Runtime estimation, meaning how long the total job processing time will be on the selected machine, provides information about the current and even future jobs. Thus, backfilling in SG will be executed smoothly, in contrast to EASY which performs it in an aggressive way. Moreover, SG has information about the jobs that have to be scheduled in order to be processed and the expected arrival time for these jobs. The expected resource utilization ratio of these future jobs is also known.

In a real grid system, runtime estimation provided by the user is far from being accurate in most cases. Thus, if the mechanism depends on the exact value of the runtime estimate, this will affect the whole scheduling quality, with costly consequences. If the runtime estimation value provided by the user is less than the actual time needed for the job to run, the processor will terminate the job. Otherwise, inaccurate and random long estimates provided by the user could lead to an unnecessarily long waiting time for the job, while it is being allocated to a free processor.

For example, if the real runtime estimate for one job is equal to five time units, but if the user estimated three time units, the processor will terminate the job when the runtime exceeds three time units. On the other hand, if the user estimated eight time units, the job must wait for a processor that can handle this size, although a free processor with six time units available that could have handled the requested job, remains unoccupied.

SG solves this issue by adapting the technique introduced by Zotkin and Keleher as in [132]. Nevertheless, as mentioned above, multiplying the real estimation time by a substantial factor such as two or more, indeed improves the performance [14, 132]. This anomaly is caused by the confluence of three factors. First, real workloads are

full of short jobs. Secondly, these short jobs benefit from backfilling more than do large jobs, as they fit more easily into the gaps. Finally, the delay affects the short jobs more than the large ones. Thus, increasing the execution time estimates provided by multiplying them by a substantial factor allows the scheduler to rearrange the jobs and backfill them in the existing gaps.

Simulations conducted in [14], showed the effect of user estimation time on performance. The experiments included three workloads and two performance metrics: response time and slowdown. The simulated mechanisms were EASY and Conservative Backfilling (CONS). Several estimation times multiplied by different factors (*f*) (2, 4, 11, 31, 101 and 301) were tested. *f* represents the badness factor, the greater *f* the less accurate runtime estimates. For example, when *f* is equal to one, this indicates full accurate estimates.

The results showed that performance improved for all the previous values, compared with the original value of runtime estimates, even if the original runtime estimate was accurate. Figures 4.4 and 4.5 based on [132] show how the original estimation time multiplied by a substantial factor is much better than the original estimation time.

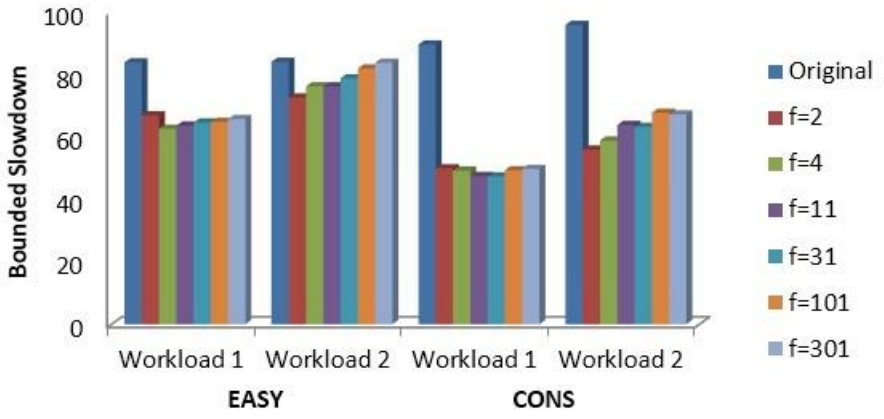


Figure 4.4. The effect of user estimated quality for slowdown [132]

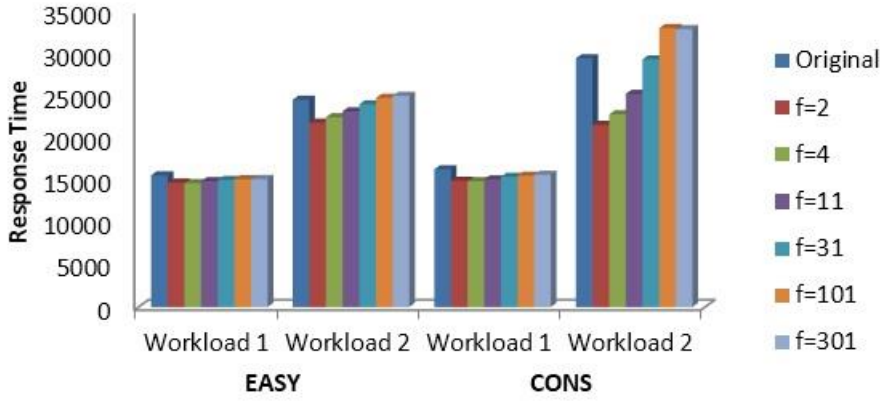


Figure 4.5. The effect of user estimated quality for response time [132]

The effect of multiplying the real time estimates by a factor can be clearly observed: much better when the real time estimates are multiplied than were the original time estimates. Even though each mechanism has its own properties, multiplying the real time estimates by a factor in order to give the mechanism more flexibility to backfill jobs always results in a better performance than the original real time estimates.

4.3 The Design of Swift Gap

The proposed mechanism, SG, increases the efficiency of the system by exploiting the idle gaps in the resources. SG is a combination of two mechanisms: Best Gap, and a local search optimization mechanism, Tabu Search (TS). When the sudden termination of a job occurs, the compression function helps to maintain the current situation of the schedule as it was before the termination happened. In the following sub-sections, SG hybridization, the compression function and detailed SG working steps are discussed.

4.3.1 Swift Gap Hybridization

To perform hybridization, the number of methods to be hybridized and the level of hybridization need to be specified. Theoretically, the number of hybridized mechanisms is unlimited, although in practice it is normally two or three.

The level of hybridization is categorized as two types: loosely coupled and strongly coupled [133]. In loosely coupled type, the hybridization level is high. In this type of hybridization, the first mechanism will finish the execution followed by the second method which optimizes the result obtained from the first mechanism. In addition, there is no interaction between the structures of the mechanisms. This type of hybridization can be observed as a chain: M1$\longrightarrow$M2, where M1 and M2 refer to the first Mechanism and the second Mechanism, respectively [133].

In strongly coupled hybridization, the methods interact with each other and thus the inner steps for each method can change. This type of hybridization is low level. It can be expressed as follows: M1 (M2), where M1 is the main mechanism that calls the secondary mechanism M2.

The selected mechanisms for SG are Best Gap and a meta-heuristic local search mechanism. The hybridization for SG is loosely coupled, whereas the flow of Best Gap and meta-heuristic local search is fully used in the hybridization. The execution type of SG is sequential. This means that when Best Gap starts running, the meta-heuristic local search mechanism will follow it to optimize the initial solution.

Two criteria have to be met to achieve the stoppage conditions. The first method stops when it finds the “best” place for the recently arrived jobs. The second method stops after a given time limit and/or a certain number of iterations. Table 4.1 addresses the hybridization specifications for SG, and the hybridization scheme is shown in Figure 4.6.

Table 4.1
SG hybridization scheme

Hybridization Criteria and Specifications	SG Hybridization Specifications
Selected Mechanisms	Best Gap and Meta-heuristic local search mechanism
Type of Hybridization	Loosely Coupled (High Hybridization)
Execution Order	Sequential Order

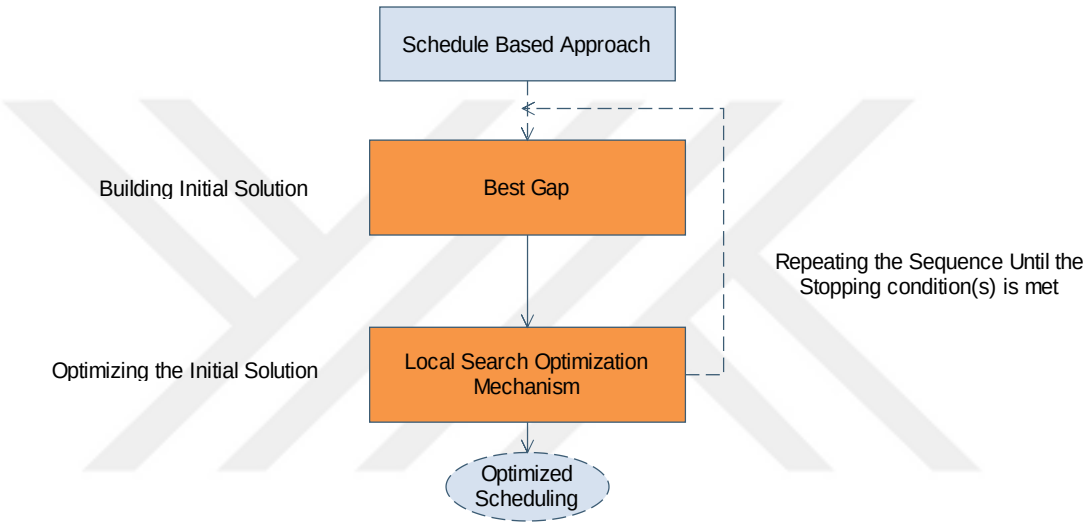


Figure 4.6. SG hybridization scheme

From Figure 4.5, Best Gap starts scheduling the jobs through by backfilling, in accordance with the runtime estimates provided by the scheduler. Once the initial solution has been generated, the optimization mechanism starts running for further improvement. In the first method, Best Gap stops when all newly arrived jobs have been placed.

The optimization starts moving (optimizing) the jobs that are ready for execution among the machines. In simple words, it just manipulates the jobs which are ready for execution. The running jobs will not be affected. When the time is over or the requested

number of iterations has been achieved, the optimization will end and the jobs will be sent for execution. Section 4.3.3 provides a detailed discussion about SG's working steps.

4.3.2 The Compression Function

As shown in Figure 4.7, a job J1 during its processing time could be terminated for an unexpected reason. The next job, J4, which is located behind the terminated job in the schedule, will jump to the free processer in order to be executed, since there is now a space in the schedule.

Before the sudden termination of J1, a backfilling decision has already been taken for J5. In the original case, if J1 is fully processed and not terminated, J5 is supposed to be backfilled and processed before J4. However, as mentioned above, J4 will jump to fill the empty space caused by the sudden termination of J1. This means that J5 is not backfilled, which contradicts the backfilling decision taken before.

Subsequently, this will force the mechanism to redo the backfilling for J5, taking extra computational time for the mechanism to handle the backfilling decision again. Since the grid deals with thousands of jobs that are waiting for execution and many resources are already reserved, the wasted computational time required to redo the backfilling is non-trivial.

To solve this, SG adopts a compression function, which maintains the schedule as it should be, even if a sudden termination has occurred. This means that when J1 is terminated, J5 will be backfilled as originally decided as shown in case 1, 2 and 3 in Figure 4.7.

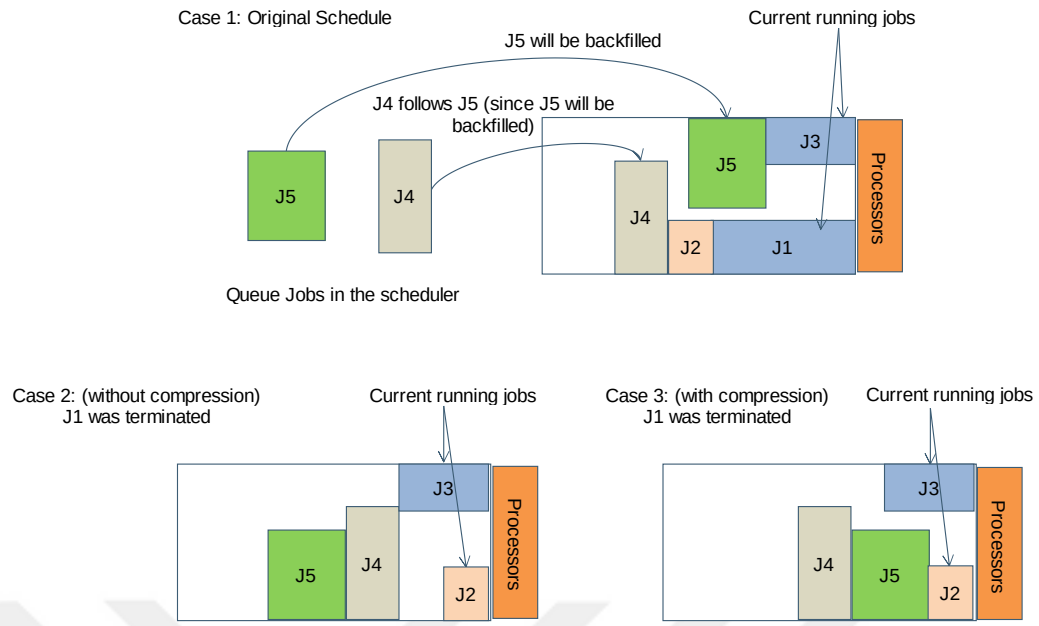


Figure 4.7. The compression function

From Figure 4.7, three different cases can be observed. The first case is the original schedule: when J1 is fully processed, J5 will be backfilled based on a previous backfilling decision and J4 comes last. In the second case, a backfilling decision for J5 has already been taken; when J1 is suddenly terminated during execution, J2 is now being processed, and J4 jumps to fill the space created by the termination of J1. J5 follows J4, as since the mechanism has already taken the backfilling decision, it will backfill J5 in another gap. This will be costly if this scenario occurs for many jobs.

The final case represents operation of the compression function. When J1 is terminated, the compression function maintains the schedule as it was before the termination. Since J5 has to be backfilled, it will come directly after J2. Simply, the compression just shifts everything to the front based on the schedule as it was before any termination or error occurred. Finally, the compression function can also be used when a job has moved from its original position as a result of optimization. Once this job is removed a gap is generated, which will be occupied by shifting all the previous

jobs forward. This also helps to reduce the waiting time for the other jobs, since the start time is earlier.

4.3.3 Swift Gap Working Steps

SG mechanism has two phases. The first is responsible for backfilling jobs by finding the earliest gap that suits each job. The solution offered in this phase is called building the initial solution. The second phase is optimization, to improve on the first solution by applying a local search mechanism. The concept is to manipulate the job's locations among the machines. Slot A denotes the default place for a job in the machine's schedule, while Slot B refers to the proposed position.

4.3.3.1 Building the Initial Solution

Algorithm 4.1 is presented to illustrate the mechanism steps.

Algorithm 4.1 **Building Initial Solution**

1. slot [A]= slot [B]=0, i =0, j=0;
2. select Job [job1,...job_n]; Select machine [$machine_1$,...$machine_m$];
3. for i,j=1 to n,m do
4. if selected machine is suitable to perform job and gap found then
5. evaluate the solution based on acceptance criterion;
6. if new solution is good move then
7. put the job into the gap and include into the initial solution slot [A];
8. end if
9. else
10. reject the move and include it in initial solution slot [A];
11. end if
12. end for
13. return initial solution (slot [A]);

In order to build the initial solution, Best Gap includes newly arrived jobs in the system, enabling reuse of the existing schedule. This is achieved thanks to the incremental approach applied in the schedule. Best Gap backfills the job in the idle space in the machine's schedule. Once the execution of Best Gap is over, it moves the job to that particular machine schedule.

To build the initial solution, the following procedure is applied. Each incoming job will be tested for all available machines (Algorithm 4.1, line:2,3). If many suitable gaps are located, the job will be moved into the one that comes first, to allow the job to be processed as soon as possible (Algorithm 4.1, line:4). If no suitable gap is currently available in all the machines' schedules, the job will be placed at the end of a schedule. Then, the mechanism moves to the next job. In order to perform the backfilling related to the job movement decision, an evaluation has to be executed. The first place for the job is Slot A, which refers to the default position for the job. The second one is Slot B, which refers to a potential new position. Moving a job to the detected gap or keeping it in the current position is based on fairness among the users (Algorithm 4.1, line:5).

The job can be moved or held in its current position based on the acceptance criterion outlined above. If Slot B is accepted, the job will be moved and this solution will be included in Slot A (Algorithm 4.1, line:7). Otherwise the job will stay in its current position and Slot A will be updated (Algorithm 4.1, line:10). This evaluation will be applied to all newly arrived jobs for all available machines. Once all jobs' evaluation is over, Slot A will be introduced as the new initial solution and it is ready for the next level, optimization. (Algorithm 4.1, line: 13).

Finally, two further cases have to be considered. The first is when the schedule is

empty, i.e. there are no jobs yet in the schedule. The second case is related to the last jobs that have not been backfilled. In the first case, the empty schedule contains a gap of infinite duration, which can be used to fit any new job. The same situation applies at the end of schedule, i.e. after the last job in a schedule, there is naturally an infinite gap that can be used for jobs that do not fit inside classic gaps. Figure 4.8 presents the flowchart for this stage.

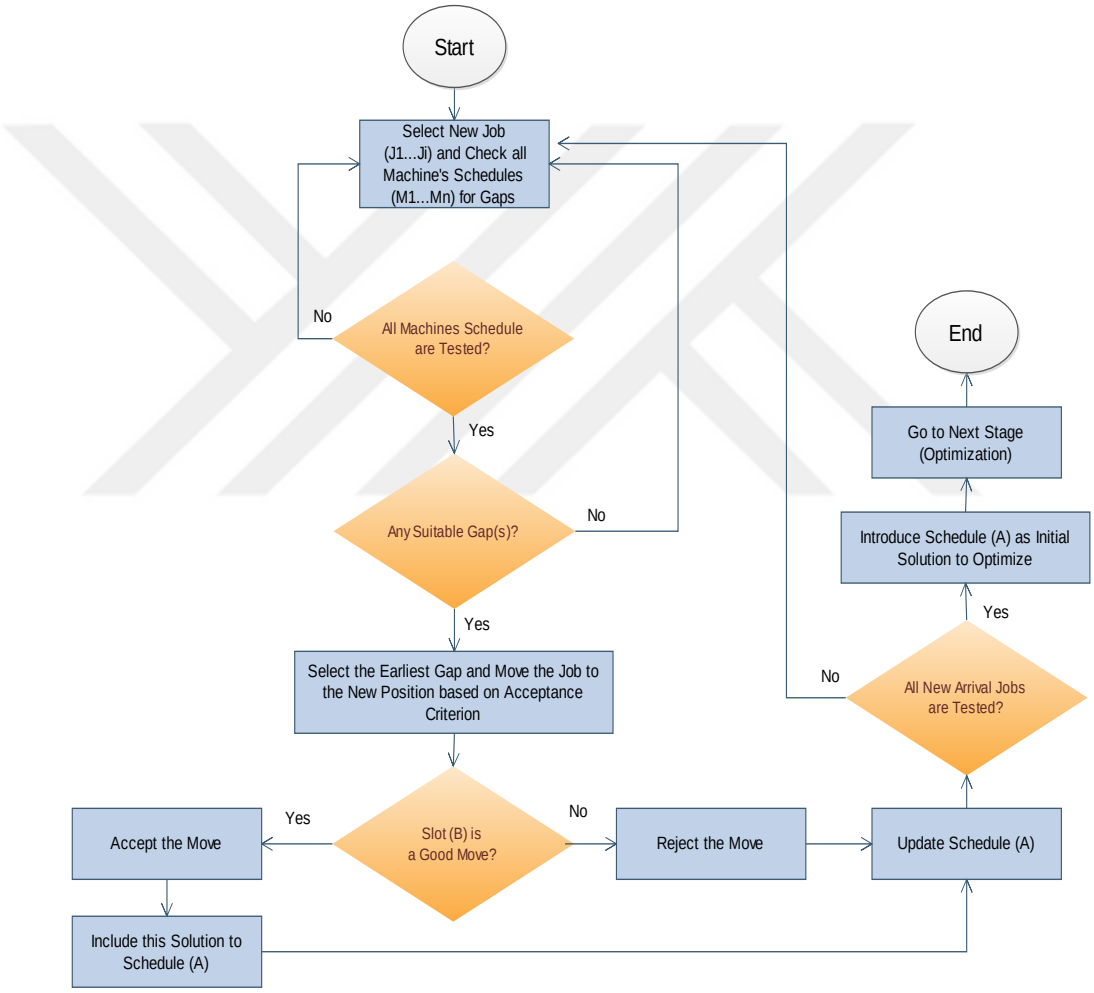


Figure 4.8. The flowchart for building the initial solution

The reason for including the Slot B solutions in Slot A, in case the Slot A solution is rejected, is as follows. When the optimization starts, it will optimize the initial solution as already explained. The initial solution has to be the best initial solution that can be

provided. Thus, all the best solutions, whether they belong to Slot A or Slot B, that were executed in the previous stage for all jobs have to be stored in one slot. This means that all accepted solutions for Slot B will be logically stored in Slot A. However, the new place for the job is still based on the Slot B solution. This procedure has to be taken to allow the next stage to optimize a set of solutions stored in one slot.

4.3.3.2 The Optimization

In the previous stage, the solution provided concerns only the jobs newly arrived in the schedule. Even though this solution exploits the gaps in the machines' schedules, it ignores the previous jobs in the schedule. Thus, many gaps can appear in the schedules. In addition, the local search meta-heuristic mechanism has extra features, which Best Gap does not have, such as historical memory and strong diversification. Thus, optimizing the initial solution will further exploit the gaps and this will lead to better scheduling [133]. Tabu Search (TS) has unique feature, which is the short memory. Figure 4.9 presents how this algorithm avoids re-optimizing the job using the Tabujob-list.

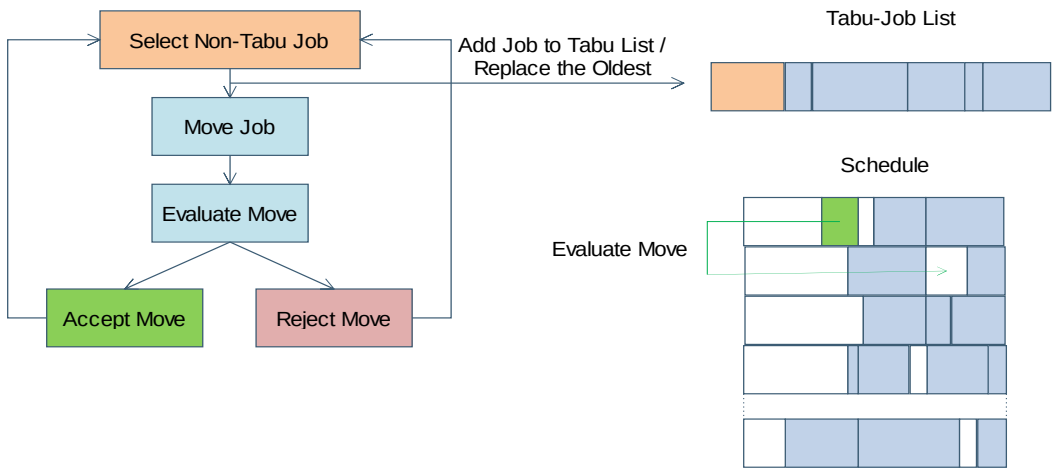


Figure 4.9. Tabu-job list

In order to perform the optimization, TS is applied [134]. Algorithm 4.2 presents the optimization stage.

Algorithm 4.2 **The Optimization**

```
1. initial solution , slot [A]= slot [B]=0, insert iteration and time limit;
2. select random Job [job1,...jobn] belongs to machine [m];
3. while (i < iterations number) do
4.       i=i+1;
5.       if all jobs tested, then tabujob-list = 0 then
6.           add selected job to tabujob-list;
7.       end if
8.      if acceptance criterion = true then
9.      remove job from slot [A];
10.     move job into earliest suitable gap and compress slot [A];
11.     include solution into slot [A];
12.     else
13.      keep the job and include in slot [A];
14.    end if
15.      if all jobs in machine schedule belong to tabujob-list then
16.        machine = used;
17.        machine selection next cycle = false;
18.      end if
19.     tabujob-list = full; remove oldest job;
20.    if optimization limit exceeded then
21.      stop;
22.     end if
23. end while
24. return slot [A];
```

Three inputs are required. The first is the initial solution, i.e. the set of solutions that will be optimized further; the second input is the number of iterations; and the third is

the time limit for the whole set of iterations (Algorithm 4.2, line:1). The optimization increases the machine usage by manipulating the position of jobs ready for execution. The jobs that are currently running will not be affected since the preemptive function is disabled in our simulation. The preemptive function enables the mechanism to modify the positions of jobs that are currently running, but enabling this function makes the execution of the mechanism complicated.

Moving the selected job to the earliest gap has three benefits. The first is exploiting the gaps that exist in the schedule for the selected machine; this improves the utilization, and subsequently will increase the number of accomplished jobs. The second benefit is the moved job will have a better chance of meeting its deadline, since it will start earlier. Once the job is selected and moved, a new gap will be generated. The generated gap will be handled using the compression function. Thus, the third advantage comes when the jobs that are behind in the schedule are compressed, which means that these jobs can start earlier too.

In order to distinguish between the optimized and non-optimized jobs, Tabu has a list, the Tabujob-list. This has a short memory in which to save a selected job that will be optimized. The list prevents the selection of a job that has already been selected for optimization. When the list is full, the oldest job is removed (Algorithm 4.2, line:19). When the iteration starts, the earliest gap in machine's schedule will be selected (Algorithm 4.2, line:2), while the selected job should not belong to the Tabujob-list.

The non-optimized job will be moved from its original location and will be included in the Tabujob-list in order to avoid reselection in the next cycle (Algorithm 4.2, line:6). If moving the job will violate the acceptance criteria, the job will be brought back to its original position. If not, the move will be accepted and the schedule will be

compressed (Algorithm 4.2, line:9,10). The solution will then be included in Slot A (Algorithm 4.2, line:11).

When all the jobs in a machine schedule have been tested, the machine will be registered as a used machine, and thus in the next cycle it will be ignored for optimization testing (Algorithm 4.2, line:15-18). When all machines have been registered as used machines, that means they have all been tested, the Tabujob-list is cleared and a new iteration starts (Algorithm 4.2, line:5). The optimization stops when the number of requested iterations is achieved or if the limit time for optimizing jobs is reached (Algorithm 4.2, line:3,20).

In the first stage (building the initial solution), the priority is to find the right place for a job; once a suitable gap has been located, the job will be moved into the gap and then the backfilling will be executed. In the next stage (optimization), the job already has a place, and optimization takes place to find a better solution if possible. However, in the optimization, the job will be included on the Tabujob-list also when the decision is made.

In the first stage, there is no need to include the job on any list that has any type of memory, because during optimization, the mechanism deals with a huge number of jobs and resources. In the first stage, the mechanism deals only with the newly arrived jobs. However, in the second stage it is important to distinguish optimized jobs from those that are not optimized yet. The acceptance criterion applied in SG are based on fairness; a job will be moved if a gap is located, and if the job fits the located gap. Figure 4.10 illustrates the optimization steps.

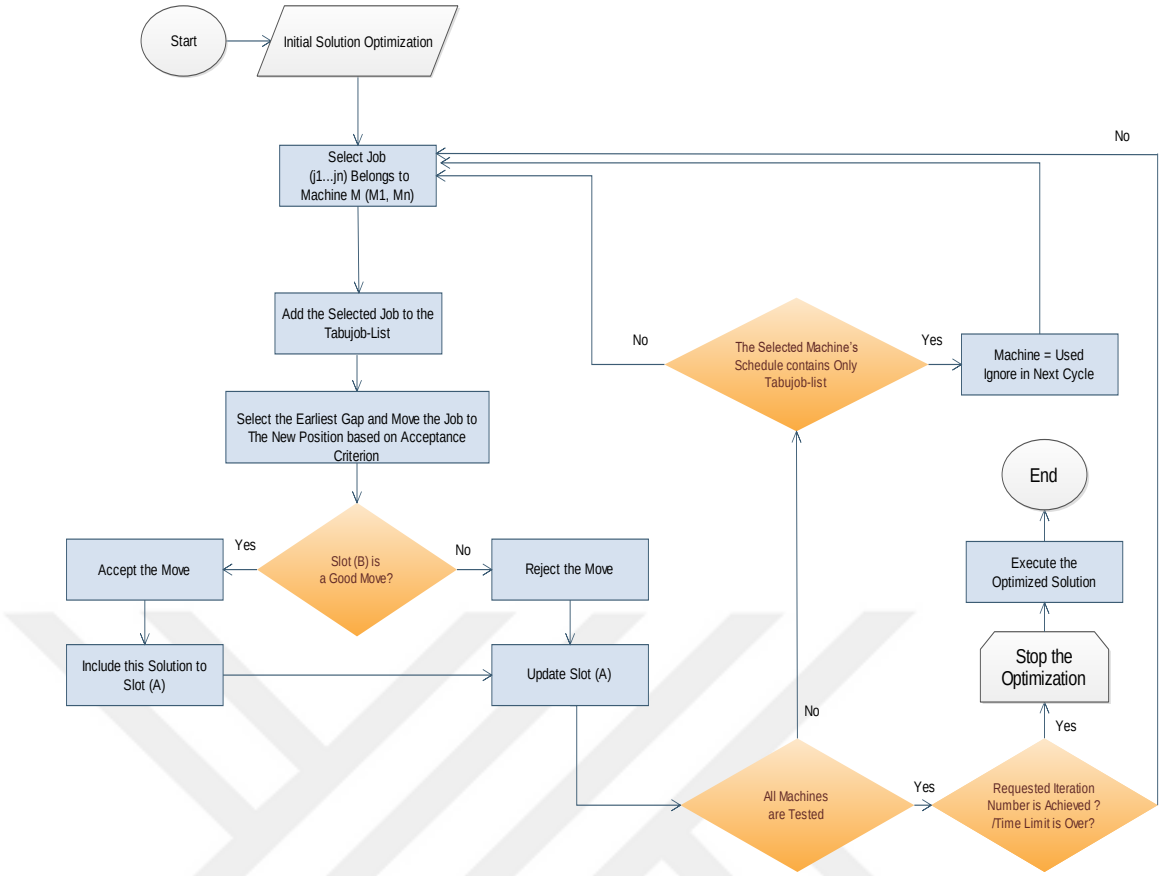


Figure 4.10. The optimization stage

4.3.3.3 Swift Gap Summary

To build the initial solution, a suitable gap has first to be found. All the newly arrived jobs are stored in Slot A incrementally. Then, when the right time slot and a suitable cluster are located, the job will be placed as follows. First, to guarantee that the job is executable, an initial test has to be done, to make sure that at least one cluster can handle it. This procedure is applied before execution.

Then, a cluster's schedule will be selected if that cluster has adequate resources and supports this type of job. If more than one schedule is available among the clusters, the selection will be based on the earliest suitable gap using the "Find First Gap" function through the "pointer". If this move is rejected, the job will be placed at the end of the schedule if no other option is found. Otherwise, the job move will be

accepted and the Slot B solution will be considered; then the solution will be stored in Slot A.

In the optimization stage, a random job in a random cluster's schedule will be selected for optimization purposes. The optimization search mechanism can move the job from one cluster's schedule to another if this move is accepted. The optimization helps the job to meet its deadline faster by exploiting the gaps in the schedule.

The first stage only considers the newly arrived jobs; previous jobs will be ignored. The reason for the gaps' existence in the optimization is as follows. The compression function was described in Section 4.3.2. This function compresses the jobs in the schedule, but it does not guarantee that all gaps have been completely removed. Algorithm 4.3 summarizes both of SG's working steps.

Algorithm 4.3 **Swift Gap**

```
1. generate initial solution for newly arrival jobs:
2. check for the gaps;
3. for all jobs and machines do
3.   if found_gaps= true then
4.   apply gap filling policy by filling the earliest gap;
5.   else
6.   schedule the job based on arrival time;
7.   apply optimizing mechanism to the generated initial result;
8.    if one of optimizing stopping conditions= true then
9.      stop;
10.    end if
11.  end if
12. end for
13. return final solution;
```

To conclude, the major difference between Best Gap, applied in the first stage, and CONS is the following. Best Gap supports fairness among users to allow them to get fair system performance, while CONS ignores this factor. The fairness of Best Gap and TS is explained in Appendix A.

4.4 Verification of Swift Gap

SG mechanism was verified using the method discussed in Chapter Three. Figure 4.11 presents a part of SG's body code after the verification process has been completed, confirming the following:

- SG has been coded correctly.

- SG implementation is free from bugs and errors.

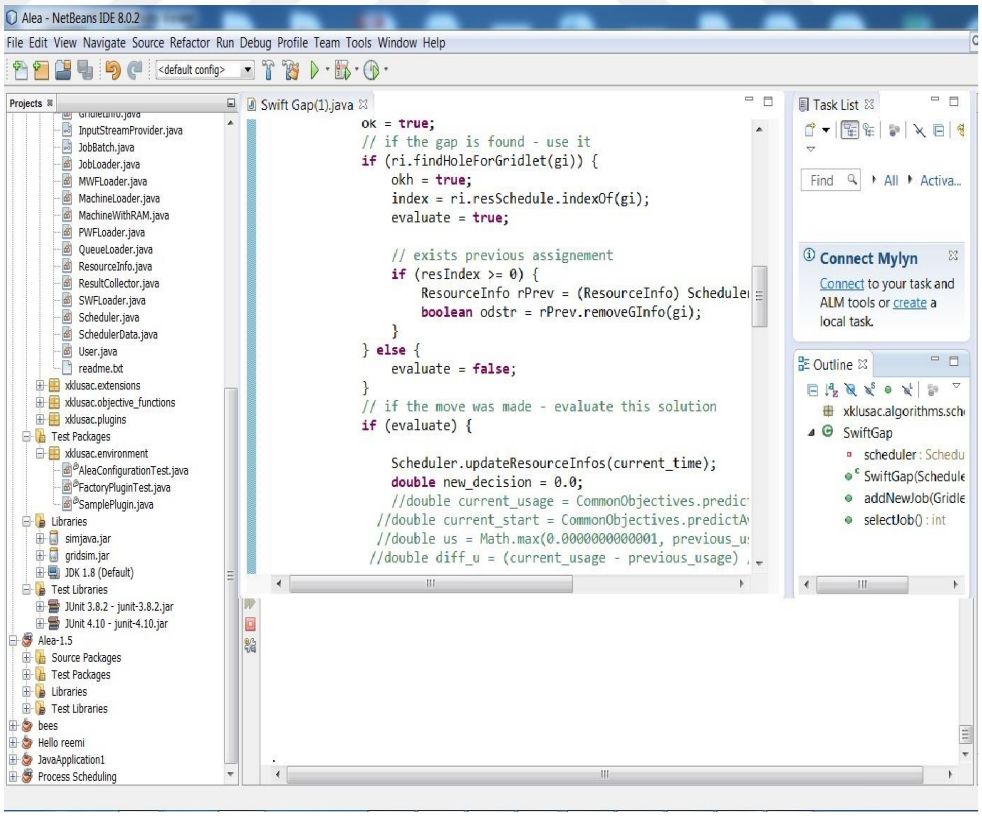


Figure 4.11. SG code in NetBeans

4.5 Validation of Swift Gap

To validate the SG mechanism, a comparison between SG and other validated mechanisms is conducted. This comparison is based on the behaviour of SG for incoming jobs in a certain workload for a particular performance metric. In other words, SG has to show similar behaviour to that of other mechanisms. Figure 4.12 shows how SG behaves when the simulated workload is "Zewura" [125] and the measured performance metric is the response time compared to CONS and ASY.

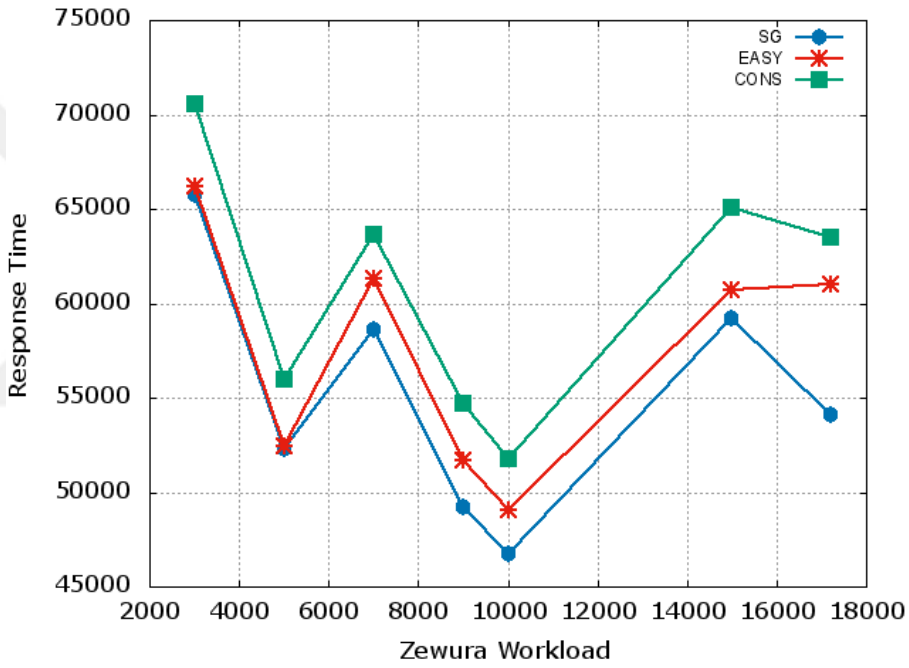


Figure 4.12. SG behaviour comparison

From Figure 4.12, the fluctuations in SG's graph is identical to the CONS and EASY graphs. The response time fluctuates according to the size of the job. For example, all three mechanisms have the lowest average response time when the number of jobs is in the ten thousands. Hence, based on the sensitivity technique [89, 90], which means that mechanisms are sensitive to the simulated input in the same way, they behave in similar way (particularly with CONS, which is more similar to the SG approach than is EASY).

As seen above, SG responds to variations in the features of the incoming jobs in the simulated workload similarly to the other mechanisms. In addition, based on the comparison [85, 91] and graphical techniques [92-95], Figure 4.12 shows that the overall behaviour of SG corresponds to CONS and EASY, which means the SG mechanism is valid. The reason behind this fluctuation in the simulated mechanisms is due to the dynamicity of the workloads. In other words, the variance among jobs' sizes and difference arrival times to the system causes this fluctuation.

4.6 Evaluation of the Swift Gap Mechanism

The evaluation of SG in this chapter is against two well known mechanisms CONS and EASY. The experiments cover three workloads, Zewura, Wagap and Meta [125], using Alea simulator [116]. The system and the details of the workloads and the measured performance metrics are presented in Section 3.5.2.4 and 3.5.2.5, respectively. Table 4.2 demonstrates the findings followed by graphical results. In the following figures, SG is the Left bar (L), EASY is in the Middle (M) and CONS is the Right bar (R).

Table 4.2
SG vs EASY and CONS in percentage ratio

Performance Metric	EASY Zewura	EASY Wagap	EASY Meta	CONS Zewura	CONS Wagap	CONS Meta
Slowdown	63%	45%	18.6%	46%	35%	17.6%
Bounded Slowdown	55.3%	32.1%	17.15%	45.3%	25%	15.5%
Tardiness	36.3%	25.4%	17.2%	48.7%	5.7%	20.1%
Waiting Time	21.55%	21.6%	16%	39.5%	5.8%	19.1%
Response Time	4.11%	0.37%	4.2%	9.25%	0.08%	5.2%

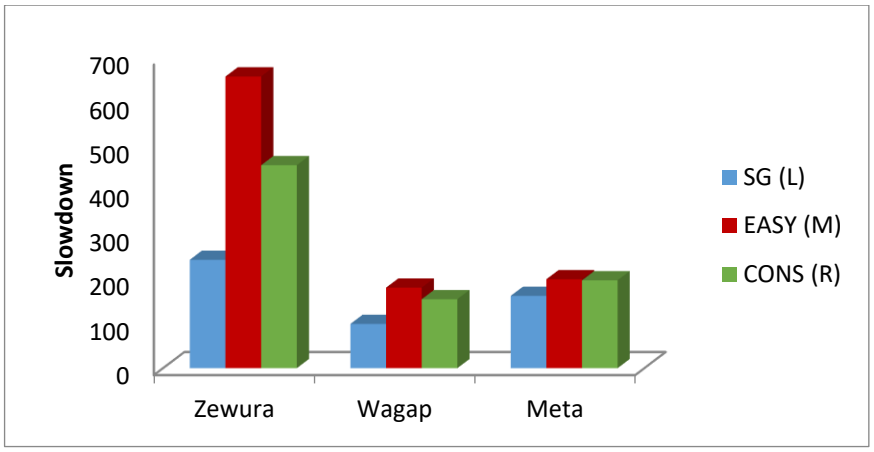


Figure 4.13. Slowdown for SG vs EASY and CONS

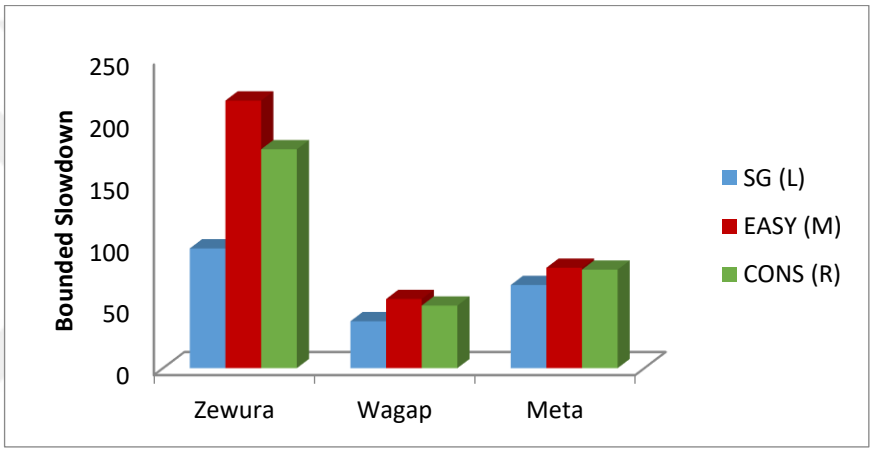


Figure 4.14. Bounded slowdown for SG vs EASY and CONS

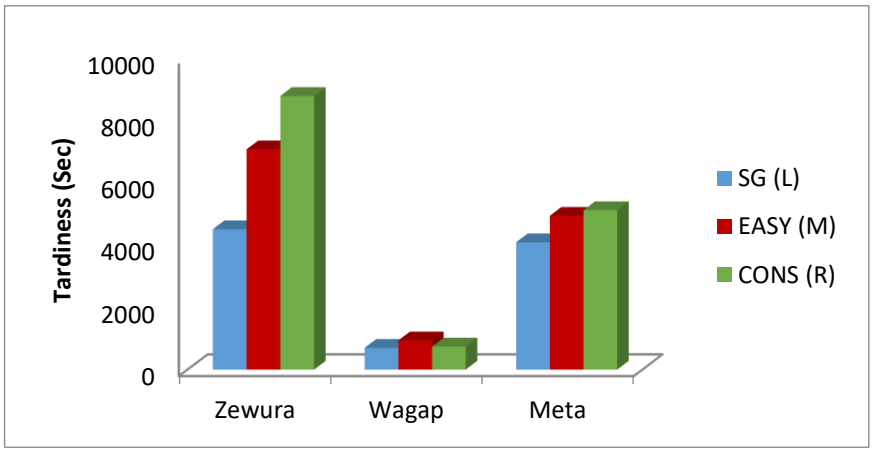


Figure 4.15. Tardiness for SG vs EASY and CONS

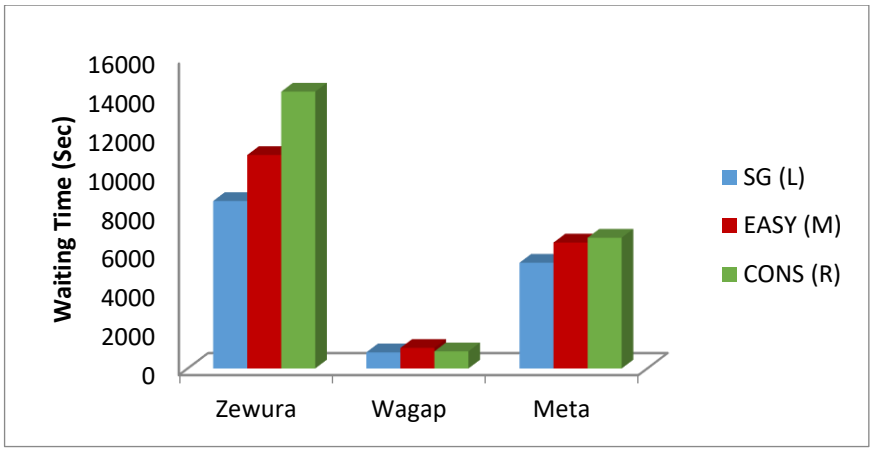


Figure 4.16. Waiting time for SG vs EASY and CONS

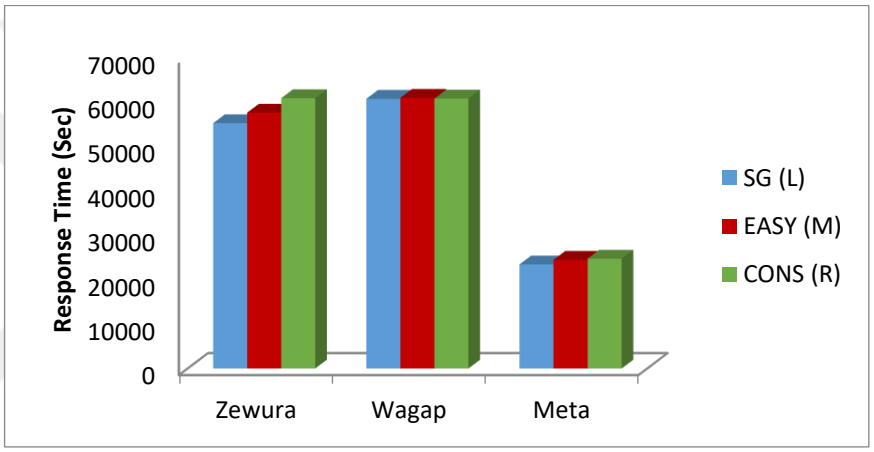


Figure 4.17. Response time for SG vs EASY and CONS

4.6.1 Results and Discussion

Table 4.2 shows the improvement ratio for the measured performance metrics in each simulated workload compared to SG. It can be observed from Table 4.2 that CONS performs better in terms of slowdown and bounded slowdown, but worse in terms of waiting time and response time compared to EASY. CONS establishes a reservation for each job, without any discrimination between the jobs; this will lead to the schedule being blocked and, in a given time, fewer backfilling jobs will be completed which will increase the waiting time, especially for the long jobs. EASY makes a reservation only for the first job in the queue, while the rest of the jobs will be backfilled without

a reservation as soon as they arrive in the system [135]. This increases the number of backfilled jobs and reduces the waiting time [136].

To analyze the performance of CONS and EASY, it is important to know that the CONS favours the backfilling of Short and Wide jobs (SW), while Long and Narrow jobs (LN) are more easily backfilled in EASY. For Short Narrow jobs (SN) and Long Wide jobs (LW), there is no specific trend. The reason behind this behavior is the way in which CONS and EASY perform the scheduling. Since CONS makes a reservation for each job, whether short or long, the long jobs have to wait for quite some time to be backfilled. Thus, short jobs are favoured in CONS [136].

On the other hand, LN can be backfilled more readily in EASY. Since the newly incoming jobs are backfilled without a reservation, LN jobs will have a better chance of being backfilled. In CONS, SW jobs have more opportunity to be backfilled and this will form a "roof" for long jobs. Thus, LN jobs suffer from backfilling in CONS, whereas they have a better chance in EASY since the spaces in the processors are not blocked by prior reservations [136, 137]. Figure 4.18 shows these job categories.

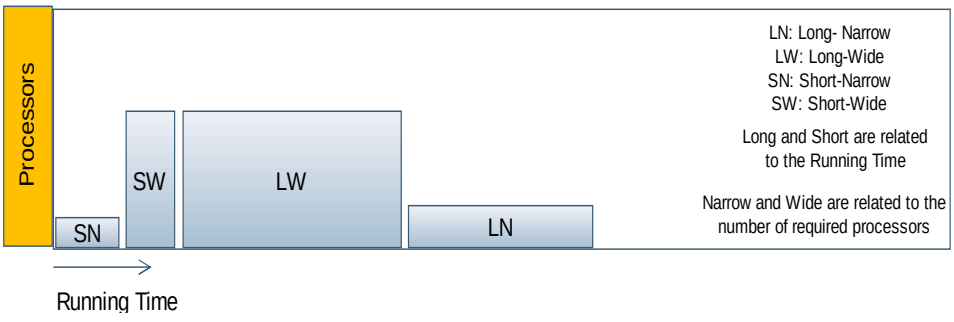


Figure 4.18. Job categorizations

CONS performs better in terms of slowdown and bounded slowdown metrics. The slowdown metric is related to the short more than the longer jobs since in the slowdown metric, the runtime (Tr) appears in the denominator [137]. Thus, the lower Tr, the

greater the slowdown will be. However, CONS backfills the short jobs more than longer jobs as discussed before, thus the waiting in the numerator for the small jobs is also small. Long jobs will take more runtime to be backfilled, hence even though the waiting time for these long jobs is huge, the runtime itself is also huge, adjusting the value of total slowdown for CONS.

The response time is more affected for the longer jobs than the short ones [137]. Since EASY favours backfilling the long jobs in, the response time is less than in CONS. However, the main question is the number of backfilled jobs in CONS that will lead to better slowdown. Thus, in order to answer this question and understand this analytical discussion, the following example is provided.

First, the main reason for this example is to clarify the confusion which can occur when calculating the slowdown for a single job by CONS and EASY, based on the fact that waiting time in CONS is greater than in EASY. If (Tr)=10, waiting time for Job (J) is 4 for EASY and 6 for CONS. Thus, the slowdown for EASY is 10+4/10=1.4, while for CONS the slowdown is 10+6/10=1.6. This simple calculation rebuts the results above that demonstrate that slowdown for CONS is less than for EASY. Therefore, more jobs have to be considered to obtain a more realistic picture.

Suppose there are 10 jobs; 6 out of 10 are short, and the rest are long. Table 4.3 states the runtime details for each job, followed by the waiting time for each job in EASY and CONS.

The runtime values are fixed for both mechanisms, while the waiting times differ. These waiting time values are based on the fact that EASY schedules both types of job, but it favours the long ones, whereas CONS backfills the short jobs faster than the

longer jobs. Thus, the total waiting time for long jobs in EASY is less than in CONS.

Table 4.3
Jobs parameters

Jobs	**J1**	**J2**	**J3**	**J4**	**J5**	**J6**	**J7**	**J8**	**J9**	**J10**
Run Time	0.7	0.5	0.6	0.8	0.9	1.2	7	8	9	10
Waiting Time (EASY)	3	2	4	2.2	6	3	1	2	3	4
Waiting Time (CONS)	0.3	0.1	0.4	0.6	0.7	1	11	13	10	15

On the other hand, the total waiting time for short jobs in CONS is less than in EASY, since short jobs have the advantage of being backfilled sooner. This example calculates the total waiting time, slowdown, waiting time and response time metrics for both mechanisms to emphasize that CONS has a higher waiting time, response time and lower slowdown than EASY, even though this may appear to be contradictory on first impression.

For EASY, the total waiting time is 28.2, the total response time is (28.2+40.7= 68.9). The slowdown for EASY is (3.7/0.7 + 2.5/0.5 + 4.6/0.6 + 3/0.8 + 6.9/0.9 + 4.2/1.2 + 8/7 + 10/8 +12/9 + 14/10 = 5.2+5+7.6+3.75+7.6+….=34.68). For CONS, the total waiting time is 52.1, which is greater than for EASY. The total response time is (28.2+53.7= 81.9). The slowdown is (1/0.7 + 0.6/0.5 + 1/0.6 + 1.4/0.8 + 1.6/0.9 + 2.2/1.2 + 18/7 + 21/8 + 19/9 + 25/10 = 1.42+1.2+1.6+1.75+1.7+1.8+….= 19.275 , which is better than for EASY.

Tardiness depends on the start time (St), tardiness equation in this research is:

$$tar = St - Pj \tag{4.1}$$

In the Wagap workload, the tardiness for CONS is less than for EASY; this is because

the jobs in this workload start earlier, while in Zewura and Meta workloads, the start time is later for CONS. To verify the tardiness results in this study, when CONS has high tardiness than EASY, the waiting time for CONS has to be greater than for EASY and the same for EASY at the other end. It can be concluded that the majority of jobs in the Wagap workload belong to the SW category; this explains why CONS performs better in terms of tardiness and, implicitly, in the waiting time compared to EASY.

SG performs better than EASY and CONS in all performance metrics in each simulated workload, as a result of the optimization applied in the second stage. However, the evaluation criteria based on the Completion Time Scheme (CTS) have not yet been applied in the previous evaluation, although SG improves the otherwise relatively poor performance of CONS and EASY thanks to the fairness and optimization techniques which are present in TS by default.

4.6.2 Swift Gap Real Time Comparison against EASY and CONS

The presentation of the evaluation in this section is based on real time comparison between SG, EASY and CONS. To highlight the actual performance of SG, the following graphs present the mean for the three workloads together. Eight different readings are captured from each workload represented on X-axis. Zewura simulates from 3,000 to 17,200 jobs; Wagap 3,000 to 17,500; and Meta 50,000 to 350,000. Each point in the following graphs is the mean of three different readings from the Zewura, Wagap and Meta workloads.

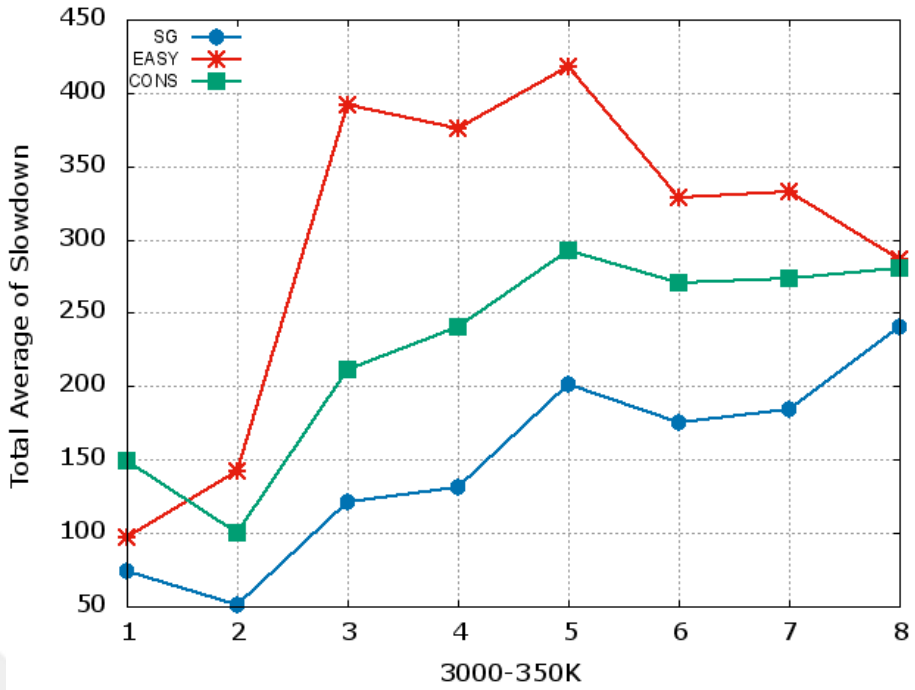
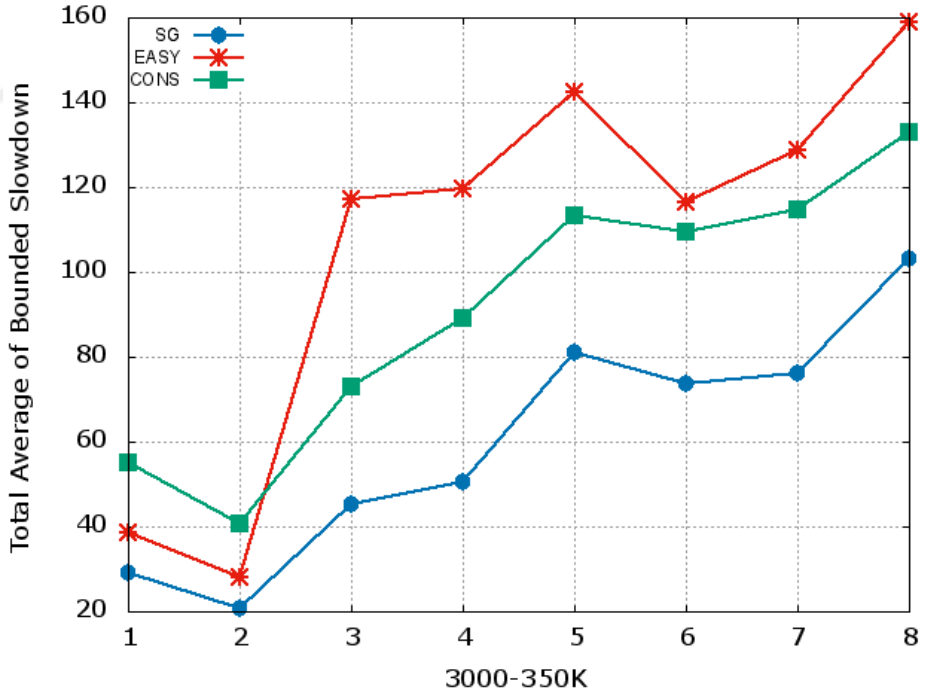


Figure 4.19. Total average of slowdown for SG, EASY and CONS



b

Figure 4.20. Total average of bounded slowdown for SG, EASY and CONS

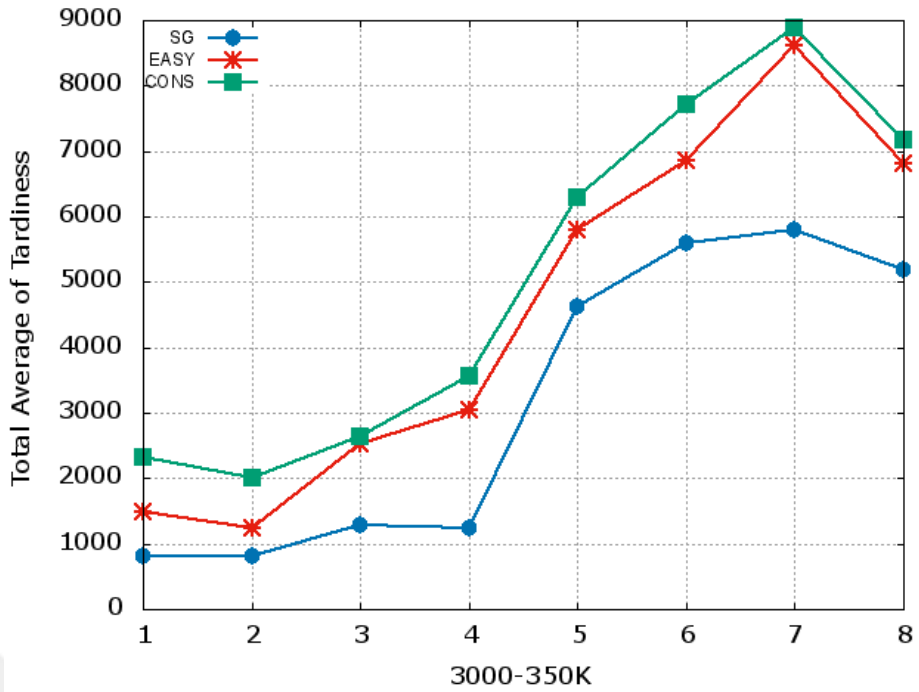


Figure 4.21. Total average of tardiness for SG, EASY and CONS

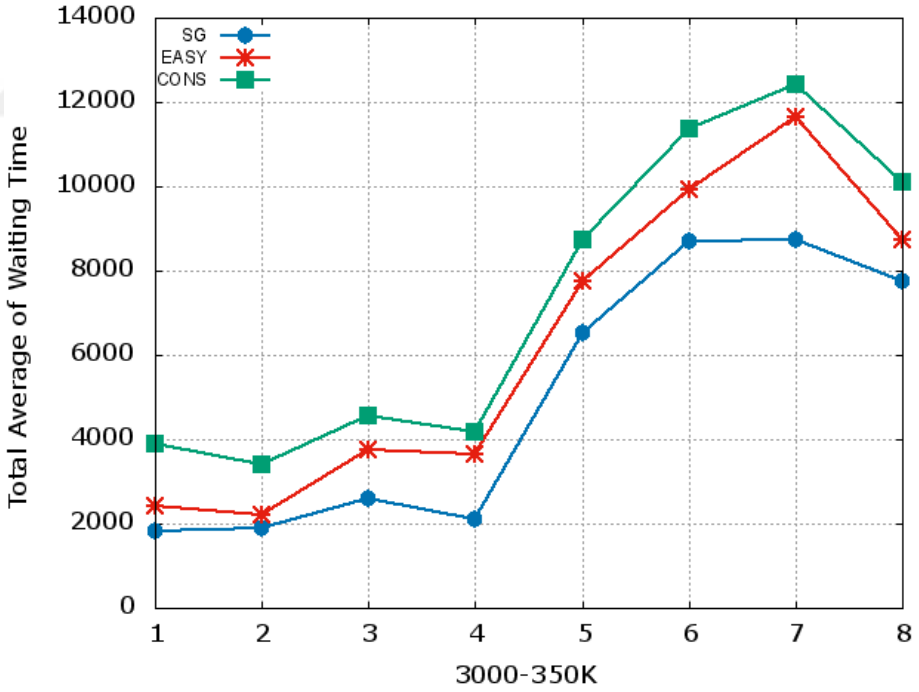


Figure 4.22. Total average of waiting time for SG, EASY and CONS

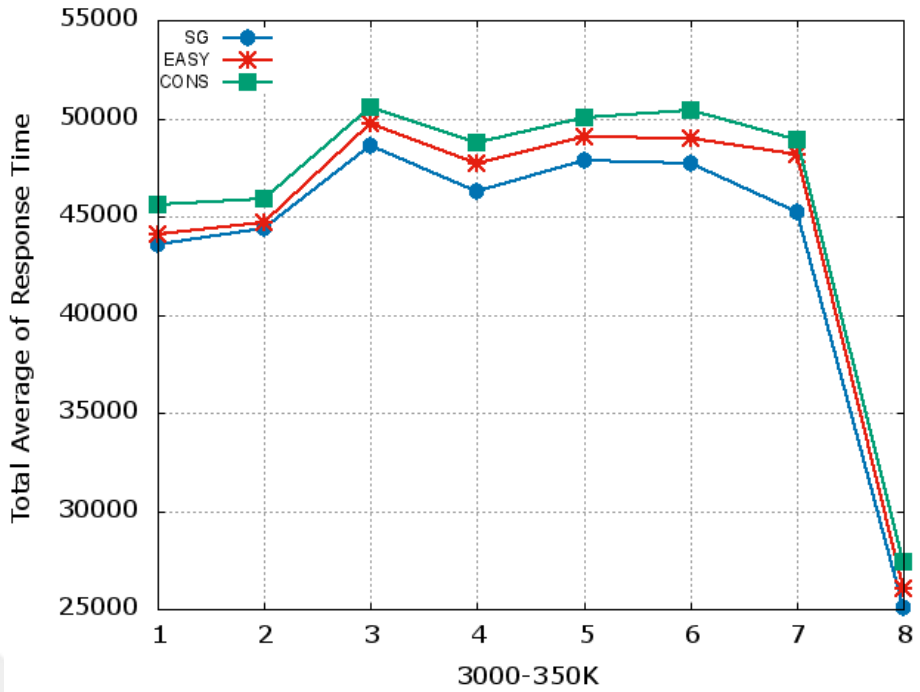


Figure 4.23. Total average of response time for SG, EASY and CONS

4.7 Summary

This chapter explained the SG mechanism for job scheduling in the computational grid using the backfilling technique. SG is the outcome of the hybridization between Best Gap and TS. It has two stages. The first stage concerns building the initial solution, achieved by placing new jobs in the earliest gaps. The next stage optimizes this solution by moving the jobs using a variety of TS's features, such as its movement and mnemonic capabilities. SG exploits the ability of planning the scheduling approach in providing the required parameters in order to schedule the jobs in advance.

The following chapter presents the proposed CTS, which further improves the performance of SG.

CHAPTER FIVE
THE COMPLETION TIME SCHEME

This chapter describes the completion time scheme (CTS), based on the weight function. The CTS shows a significant improvement in performance, which in turns improve the Quality of Service (QoS). Reducing the completion time is based on reducing the start time and the processing time for the job. This chapter is organized as follows. Development of the scheme is discussed in Section 5.1, and its evaluation and results are discussed in Section 5.2. Section 5.3 presents the influence of the CTS on the simulated performance metrics. Section 5.4 concludes the chapter.

5.1 The Completion Time Scheme

The original SG performs well. However, the backfilling and optimization are based only on fairness. Performance can be improved by guaranteeing that any backfilled job will start as soon as possible, while maintaining the efficiency of the resources. The scheme tries to find which position (A or B) for a selected job has the shortest completion time. The shortest time means that less time is taken to accomplish the selected job from start to finish. In the developed CTS, there are two inputs, Slot A and Slot B. Slot A refers to the current position for the job, and Slot B to the proposed position.

Slot A and Slot B will be compared based on the weight function. The weight function is used in order to determine the decision when there are multiple criteria to be considered [96]. If several criteria need to be synchronized, the sum of the variables has to be included in the weight function [138].

For some criteria, the lower the computed value of the variable, the better the

performance metrics. For others, a lower value means deterioration in the performance metrics. For instance, less response time means less delay (improvement) which is represented in Equation (5.1), but less resource utilization increases the number of delayed jobs (deterioration) which is represented in Equation (5.2). To compute the weight function for a metric, the following equations are used [138].

$$Weight_{\ Metric} = \frac{A - B}{A} \tag{5.1}$$

$$Weight_{\ Metric} = \frac{B - A}{A} \tag{5.2}$$

Where A refers to the value of the computed metric when the job is located at position A, i.e. the Slot A solution, while B refers to the value of the computed metric when the job is located at position B, i.e. the Slot B solution. Switching between A and B in the numerator and fixing the denominator in Equation 5.1 and Equation 5.2 enables mixing both $\text{Weight}_{\text{Metric}}$ in one total weight equation as given below, to reach the final decision between Slot A and Slot B when multiple criteria have to be considered:

$$Total\ weight = \ Weight\ metric_1 + Weight\ metric_2 + \cdots Weight\ metric_n \tag{5.3}$$

As discussed above, improvement in some criteria means a smaller value for the computed performance metrics, while in others, the greater the computed values the better the performance metric. Thus, if two different criteria have to be computed in one total weight, both Equations 5.1 and 5.2 have to be used. Otherwise, either Equation 5.1 or Equation 5.2 has to be computed when the $\text{Weight}_{\text{Metric(s)}}$ belongs to the same category. In this research, two criteria are implemented using weight function. Equation (5.4) demonstrates the total weight function applied in this study.

$$Total\ weight = \ Weight\ metric_1 + Weight\ metric_2 \tag{5.4}$$

For instance, if the total weight concerns only the delay metrics, such as response time, only Equation 5.1 is used. If the total weight concerns only utilization, only Equation 5.2 is used. Finally, if the total weight concerns both previous concepts at the same time, both Equations 5.1 and 5.2 have to be computed.

Moreover, based on Equations 5.1 and 5.2, when the total weight is smaller than zero, (Total weight ≤ 0), then the Slot A solution is better than Slot B and the job movement will be rejected. If the total weight is greater than zero (Total weight > 0), then the Slot B solution is better than Slot A and the job movement will be accepted. The completion time equation is [98]:

$$Cj = Sj + Pj \tag{5.5}$$

where Cj is the completion time for a job, i.e. the time required to execute the job execution, Sj is the start time, which refers to the actual time the job begins to be processed, and Pj is the CPU processing time for the job. From Equation 5.4, it is obvious that to reduce Cj, Sj and/or Pj has to be decreased. Thus, the implementation of the CTS in this research will cover the two variables in the previous equation.

5.1.1 Improving Completion Time through Start Time

From Equation 5.4, selecting the best Sj refers to the earliest time that a job can be handled by the resource. This will contribute in reducing Cj, which in turn improves the performance of the scheduling mechanism. In the first stage (building the initial solution) and second stage (optimization) of SG, the earliest gap will be selected and compared with the current position of the job. Algorithm 5.2 illustrates the selection of best Sj based on the weight function proceeded by Algorithm 5.1 which presents how the schedule is checked before Algorithm 5.2 starts.

Algorithm 5.1 **Checking the Schedule**

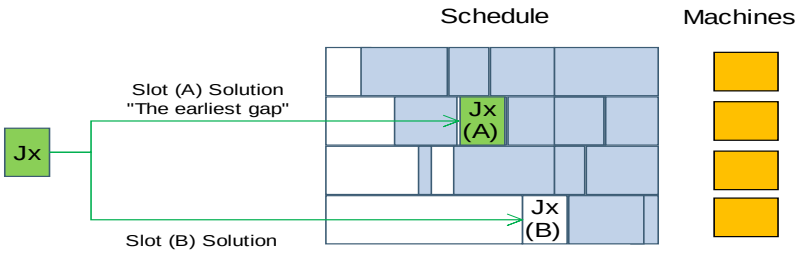
1. [machine's schedule 1, …… machine's schedule n];
2. for (i < n) do
3. if last job finishing time in first detected machine's schedule < earliest gap start time then
4. proceed to weight function (Algorithm 5.2);
5. else if
6. place the job into the earliest gap;
7. end if
8. end for

Algorithm 5.2 **The Completion Time Scheme (Start Time)**

1. get the start time for Slot (A);
2. get the start time for Slot (B);
3. weight $_{\text{start time}} = \frac{As-Bs}{As}$
4. if weight ≤ 0 then
5. reject the move and select Slot (A) as a solution;
6. else
7. accept the move and select Slot (B) as a solution;
8. end if

However, the earliest gap does not always guarantee the best Sj. In some cases, the earliest gap may be located at the back of the schedule, i.e. farther away than the processors, compared to the back of another machine's schedule. This happens if the end of at least one machine's schedule has more vacancy than the earliest gap. Unfortunately, the backfilling mechanism will not backfill the job at the end of schedule since it keeps the tail only for jobs that do not fit in the schedule or when the

acceptance criterion rejects the proposed place. The backfilling mechanism in the first stage tends to fill the gaps rather than placing the job at the end of the tail, although this procedure may delay the job. Figure 5.1 clarifies the selection of best time.



If As = 9, Bs = 8
Weight start time = A-B/A = 9-8/9= 1/9 = +0.11
Slot B is better (job will be placed at end of schedule)

As: (Start time in A position)
Bs: (Start time in B position)

Figure 5.1. Reducing the completion time via selecting the best start time

In order to perform the previous action, the earliest gap has to be detected through the "Find First Gap" function. Then the mechanism scans the end (tail) for each machine's schedule. If the tail of the first machine's schedule start time detected is less than the earliest gap, i.e., $end\ machine's\ schedule\_time\ <\ earliest\ gap\_time$, the weight function will be applied. Otherwise, the earliest gap will be selected straight away.

However, a question that might arise is, if one machine's schedule time is less than the earliest gap, why will the mechanism proceed to the weight function? Since the grid is extremely dynamic, there is the possibility that a job (ahead) may be terminated in the earliest gap's schedule; hence the Sj for the earliest gap will move forward and consequently it may be better than the current end machine's schedule. Therefore, the weight function confirms if the end machine's Sj is still less than the earliest gap or

not.

5.1.2 Improving the Completion Time through the Processing Time

There are two main factors that affect the processing time (runtime) for a job: the size of the job and the machine capacity. The processing time Pj in Equation 5.4 can be calculated based on Equation 5.6.

$$Pj = \frac{Jsz}{Rcap} \tag{5.6}$$

where Jsz is the size of the job and $Rcap$ is the resource capacity. The size of the job is a fixed value determined by the task sent by the user. The resource capacity is also a fixed value that depends on the available resources. However, Pj can be reduced if a faster machine is selected. The faster the machine, the more resource capacity there is in the denominator, and thus the processing time is reduced. The Algorithm 5.3 presented clarifies the selection of the fastest machine.

Algorithm 5.3 **The Completion Time Scheme (Processing Time)**

1. get the weight machine usage for Slot (A);
2. get the weight machine usage for Slot (B);
3. weight $_{\text{weight machine usage}} = \frac{\text{Bu}-\text{Au}}{\text{Au}}$
4. if weight ≤ 0 then
5. reject the move and select Slot (A) as a solution;
6. else
7. accept the move and select Slot (B) as a solution;
8. end if

In order to apply this concept to the existing resources, the job has to be moved to a

faster machine if one is available, which is where the machine utilization weight comes in. The machine utilization weight refers to the fastest machine available in the schedule for the selected job, i.e. the amount of utilized CPU operations [139]. This concept differs from the classical utilization of the machine, where the most important thing is to keep the machine fully utilized in order to exploit any idle time that can be detected when a gap exists in the schedule.

The classical utilization of resources works very well when it concerns exploiting the gaps in the schedule. However, this concept ignores the fact that not all the resources have the same computational power in the grid. For instance, Machine 1 is three times faster than Machine 2. Machine 2's schedule has a suitable gap to execute the backfilled job, but Machine 1's schedule has no available gap for the time being. Thus, based on the classical utilization, Machine 2 will be selected since it has idle CPU time. However, in such a case, implementing the classical resource utilization regardless of the resources available can be debatable.

Since Machine 1 would process the job three times faster, while Machine 2 could start earlier although the execution time might be longer. Hence, the job could be completed earlier on Machine 1, even though Machine 2 has an earlier suitable gap. Figure 5.2 presents the when the concept of machine weight utilization is applied.

Since SG takes the schedule-based approach, as discussed in Section 4.1, the computational power for the shared resources is known. Thus, moving the job based on machine speed is possible here. To compute the value for the weighted machine, the "predict machine usage" function is applied. This function calculates the execution time for the job on both selected machines and then refers to the fastest machine based

on the weighted machine speed concept. Figure 5.3, clarifies the selection of the fastest machine.

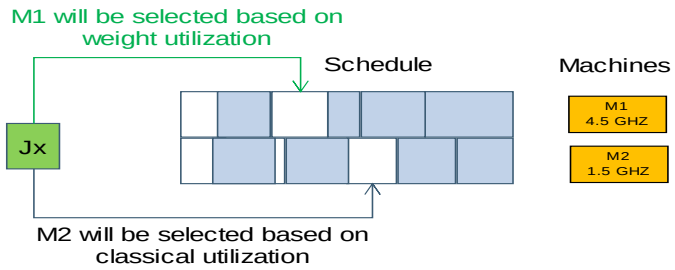


Figure 5.2. Weight utilization vs Classical utilization

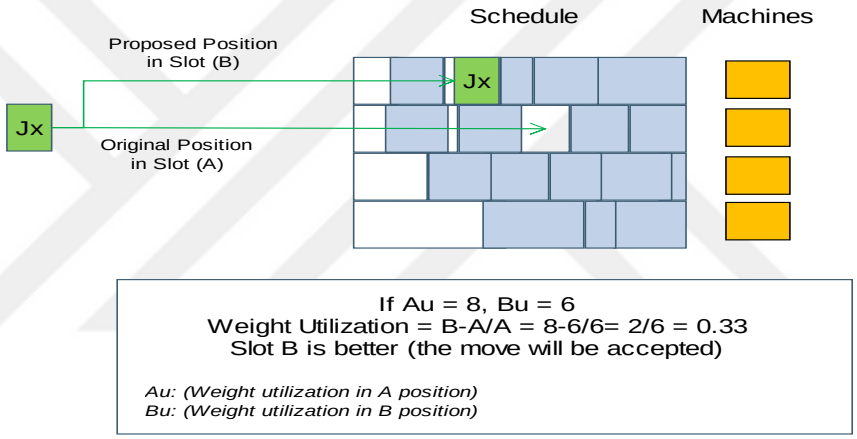


Figure 5.3. Reducing the completion time via selection of the fastest machine

5.2 Evaluation of the Completion Time Scheme

The purpose of this evaluation is to show the additional improvement when the CTS is embedded, comparing the performance with Swift Gap (SG). The setup of the full experiment is presented in Section 3.5.2.4. It is important to indicate that CTS runs dependently on SG. In the following evaluation, it is referred to the column as CTS rather than SG-CTS, since the integration between SG and CTS will be explained in the next chapter. Table 5.1 shows the improvement in performance metrics in each simulated workload when the developed CTS is applied followed by graphical results where CTS is the Left bar (L) and SG is the Right bar (R).

Table 5.1
CTS improvement ratio vs SG percentage improvement ratio

Performance Metric	Zewura Workload	Wagap Workload	Meta Workload
Slowdown	8.8%	32.2%	2.15%
Bounded Slowdown	12.2%	32.6%	0.88%
Tardiness	3%	45.35%	4.1%
Waiting Time	3.1%	36.8%	3%
Response Time	3.58%	2%	14.6%

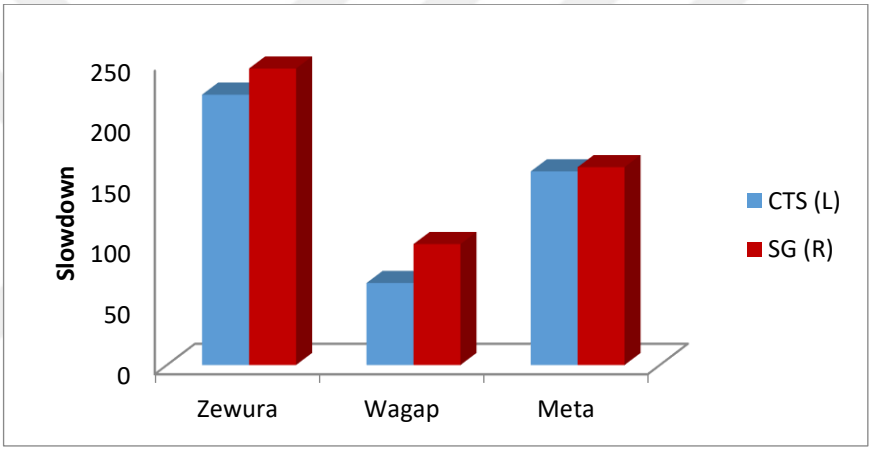


Figure 5.4. Slowdown for SG and CTS

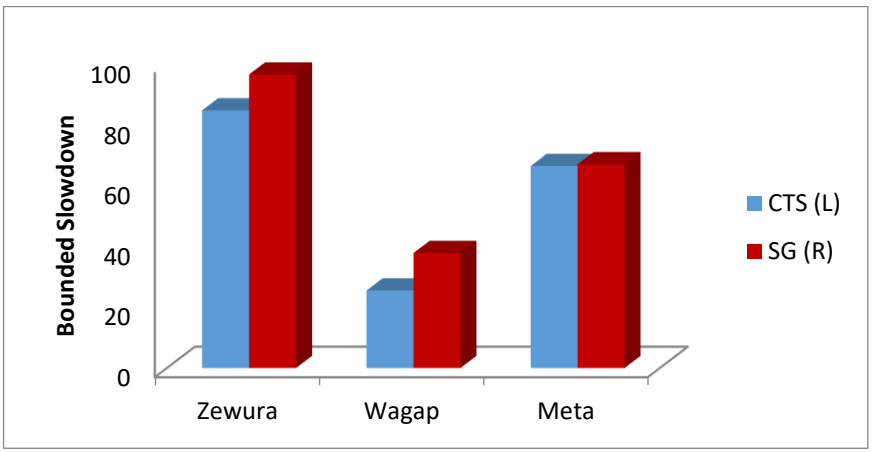


Figure 5.5. Bounded slowdown for SG and CTS

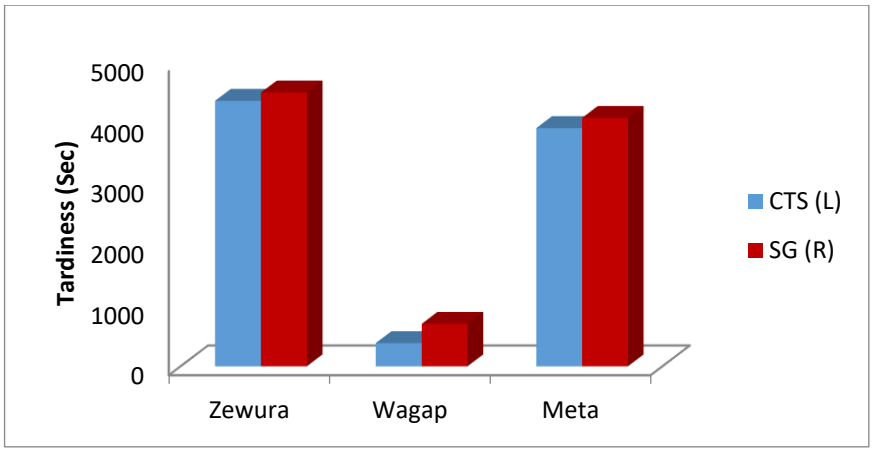


Figure 5.6. Tardiness for SG and CTS

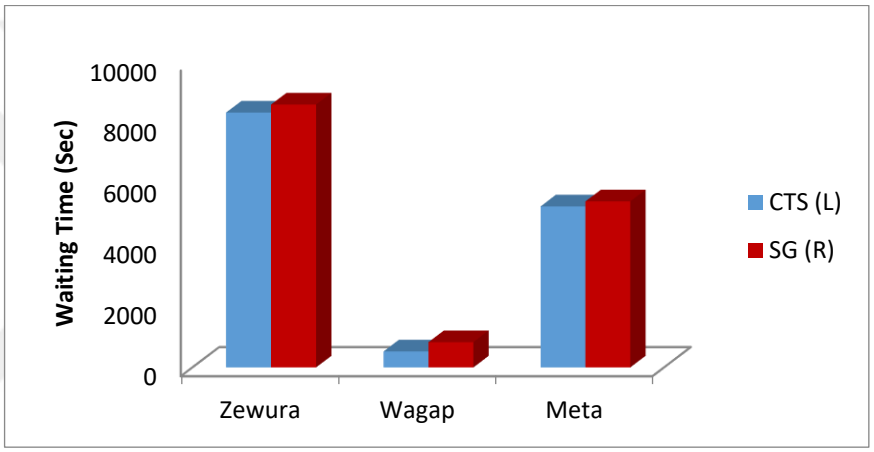


Figure 5.7. Waiting time for SG and CTS

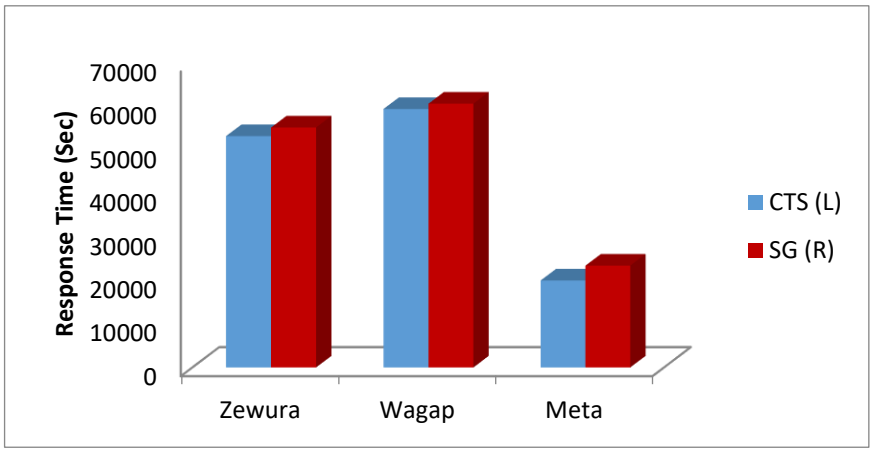


Figure 5.8. Response time for SG and CTS

5.2.1 Results and Discussion

From Table 5.1, it can be observed that CTS outperforms SG alone in all performance metrics for all simulated workloads. The CTS improves especially for the metrics where intensity is involved, explaining why it improves the slowdown, bounded slowdown, tardiness and waiting times more than response time.

The waiting time, as shown in Section 3.5.2.4 is the difference between the start time for the job and the submission time to the system. Since the start time is involved in improving the total completion time through the weight function, as shown in Section 5.1.1, the total waiting time has to be reduced after applying the CTS.

The improvement ratio in the response time is less than in the other simulated metrics. The response time is the summation of total runtime and waiting time for the job, and the developed CTS involves both the waiting time and the runtime. The CTS tries to reduce the total runtime, i.e. the processing time, by selecting the faster machine as described in Section 5.1.2. However, the runtime parameter is subjected to other factors, as shown in Figure 5.9.

Figure 5.9 explains the reason behind the smaller improvement ratio in the response time metric than in the other metrics. There are many factors and sub-factors that affect the runtime for the job. Except for the weighted machine speed, which is handled by the developed CTS, all other factors and sub-factors are subject to the workload or to the running application.

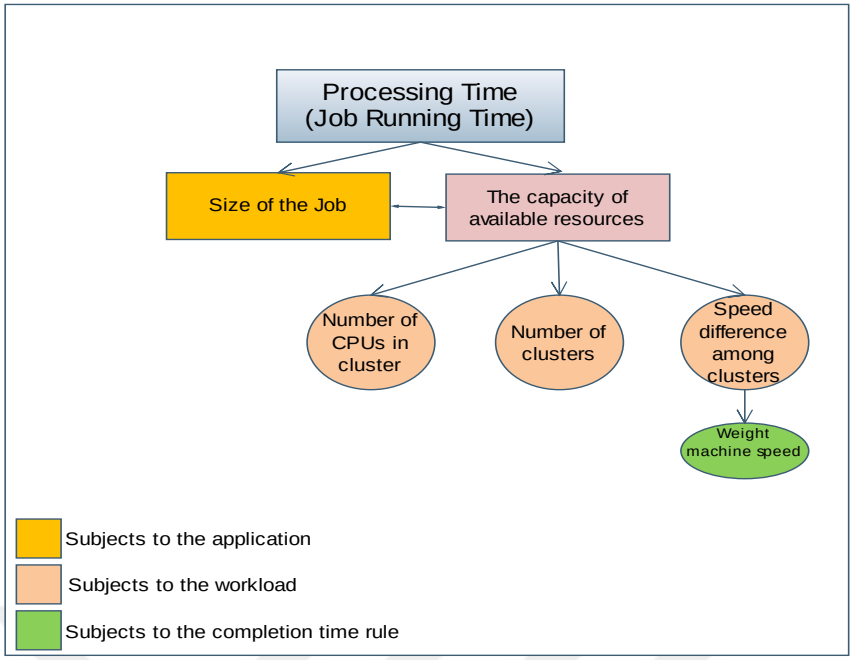


Figure 5.9. Run time factors

The size of the job in the computational grid does not reflect the volume aspect; the job itself is an object [140]. However, the size of the job from the perspective of an application that runs in a computational grid environment refers to the complexity of the job. Complex jobs demand additional computational power besides time [141]. Since this research is conducted for High Performance Computing applications (HPC), this means the running jobs are highly complex, which will increase their total runtime.

The number of computing clusters, the number of CPUs in each cluster, and the specifications of the clusters are subject to the real trace captured from the real workload. In this research, it was decided that the simulations conducted use traces captured from real workloads. Therefore, the capacity of the resources is embedded in the simulated workloads. Hence, the calculated runtime for jobs is related to the resources embedded in the workload file.

However, the specifications of the resources can be modified in the simulated

workloads, although any applied modification related to the features of clusters will reduce the integrity of the workload; thus, the specification of the clusters remains untouched. However, the response time has the least share in the improvement ratio after applying the CTS, but even 2% improvement should not be underestimated. Even though thousands of jobs are simulated in this research, in a real grid system for HPC applications the number of jobs running at any one time may exceed millions, since the real grid system runs for a long time and may last for years [8]. Therefore, even a low ratio can make a difference in the QoS.

The total tardiness is the difference between the total completion time for the jobs and the corresponding due date for each job. The corresponding due date value in the conducted simulation is considered as $2 \times running\ time\ (Tr)$ as shown in Section 3.5.2.4. Thus, the expression of tardiness in the conducted simulation in this study is:

$$tar = (St + Pj) - 2 \times Running\ Time\ (Tr) \tag{5.7}$$

where, $2 \times Running\ Time$ is the corresponding due date, St is the start time and Pj is the processing time. The runtime for the job reflects the Pj for the job, since the runtime is equal to the actual time taken by jobs being executed in parallel on the resources [126]. Based on this definition, the runtime (Tr) expression can be written as follows:

$$Tr = Completion\ time - St \tag{5.8}$$

$$Tr = (St + Pj) - St \tag{5.9}$$

$$\Rightarrow Tr = Processing\ Time \tag{5.10}$$

Based on expression (5.10), the tardiness equation can be written as follows:

$$tar = (S\,t + Pj) - 2 \times Pj \tag{5.11}$$

$$tar = St - Pj \tag{5.12}$$

The Equation 5.11 is the tardiness expression in this research; therefore, the massive improvement in this matrix does make sense, since the effect of the runtime factors and sub-factors is subtracted from the start time. This also justifies the fact that the improvement in tardiness is the best among all the other metrics. The waiting time also shows significant improvement since it is related to start time only, and the runtime is not included in this metric.

The slowdown and bounded slowdown also show significant improvement. The slowdown is the response time for jobs normalized by the total runtime. The slowdown metric is freer from the effect of out-of-control factors and sub-factors than the response time metric, as discussed earlier. Although the nominator in the slowdown metric is the response time, which includes the runtime, the runtime also appears in the denominator. This reduces the runtime effect and justifies the greater improvement in this metric compared to the response time metric.

5.2.2 Completion Time Scheme Real Time Comparison

The following comparison shows the performance of the SG mechanism with and without the CTS in a real time scenario. The method used in this comparison was explained in Section 4.6.2.

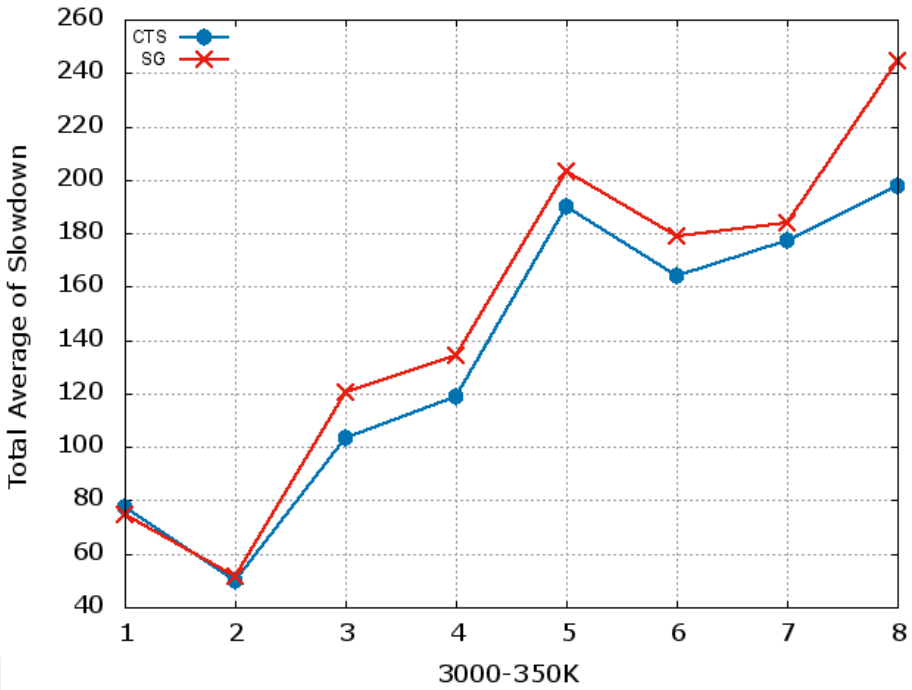


Figure 5.10. Total average of slowdown for SG and CTS

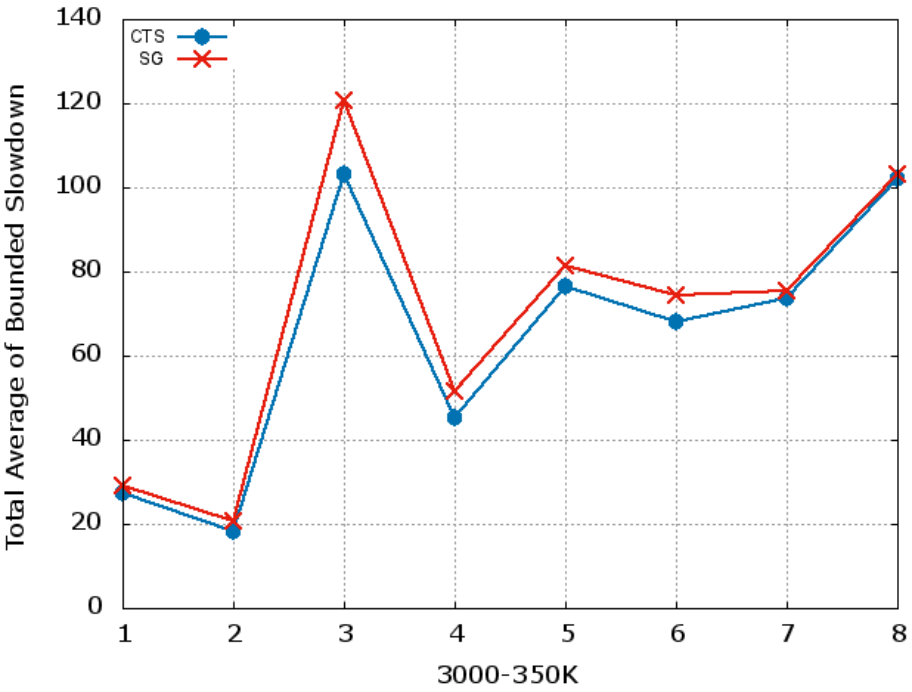


Figure 5.11. Total average of bounded slowdown for SG and CTS

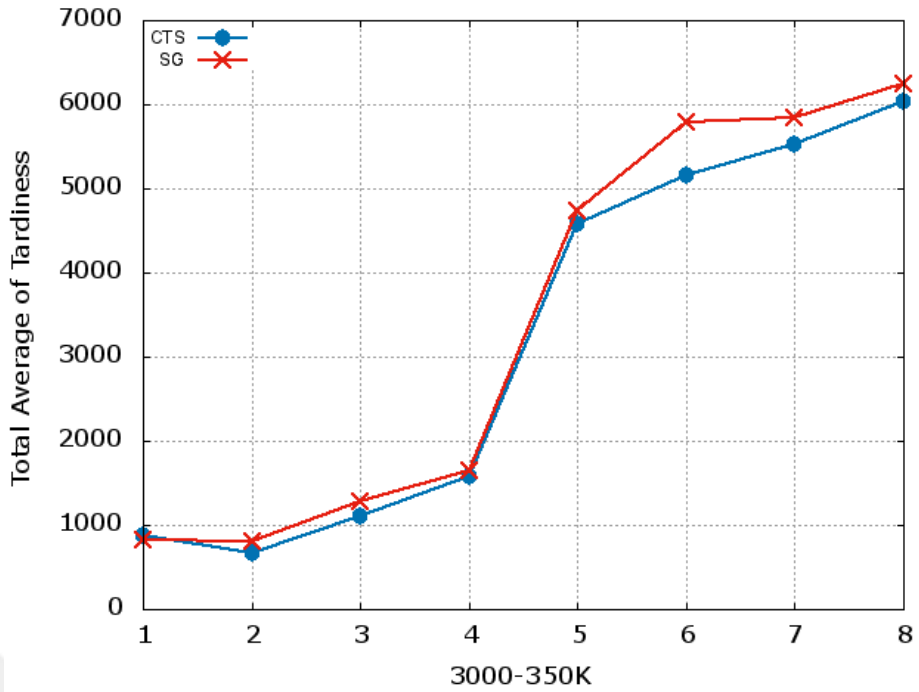


Figure 5.12. Total average of tardiness for SG and CTS

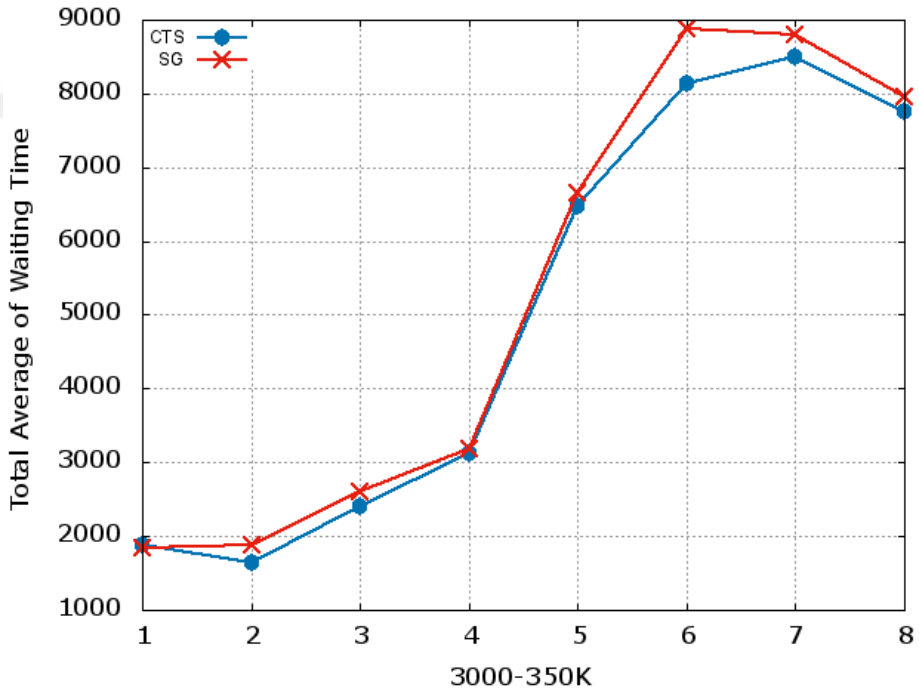


Figure 5.13. Total average of waiting time for SG and CTS

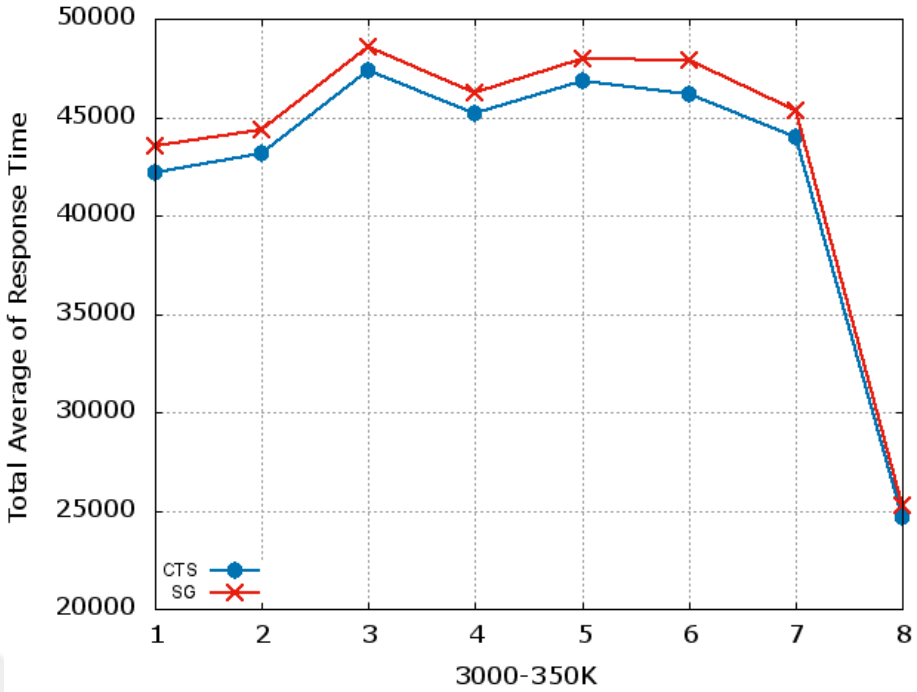


Figure 5.14. Total average of response time for the SG and CTS

The results in Table 5.1 demonstrate that the developed CTS improves the performance of SG in all the simulated workloads, especially in Wagap. The Wagap workload contains the highest ratio of short jobs, compared to the Zewura and Meta workloads. The definition of backfilling is to move the short jobs forward to exploit the idle time in the resources and improve the total utilization of the system [14, 15].

Long jobs can be backfilled as well; however, the short jobs are easier to backfill since finding a reservation for them is smoother [135]. Based on this definition, the number of backfilled jobs in Wagap has to be high; and only backfilled jobs benefit from CTS. Since the majority of jobs in Wagap are backfilled, the CTS will be applied more in this workload than the others. Therefore, the effect of the CTS clearly appears in the Wagap workload.

5.3 The Effect of Completion Time on the Quality of Service

Figure 5.15 clarifies the relationship between the simulated metrics, which present the QoS and the completion time factor in this research. Both slowdown and bounded slowdown are affected by the start time negatively, compared to runtime. This is due to the existence of the runtime in the denominator, as discussed before, making its impact stronger than the start time, which only appears in the numerator. Equation 5.16 clarifies this:

$$sld = \frac{Tr + Tw}{Tr} \tag{5.13}$$

$$sld = \frac{Tr}{Tr} + \frac{Tw}{Tr} \tag{5.14}$$

$$sld = 1 + \frac{Tw}{Tr} \tag{5.15}$$

$$sld = 1 + \frac{St - Sb}{Tr} \tag{5.16}$$

where Tr is the runtime, Tw is the waiting time, St is the starting time and Sb is the submission time. Figure 5.15 shows the relationship between the performance metrics and completion time.

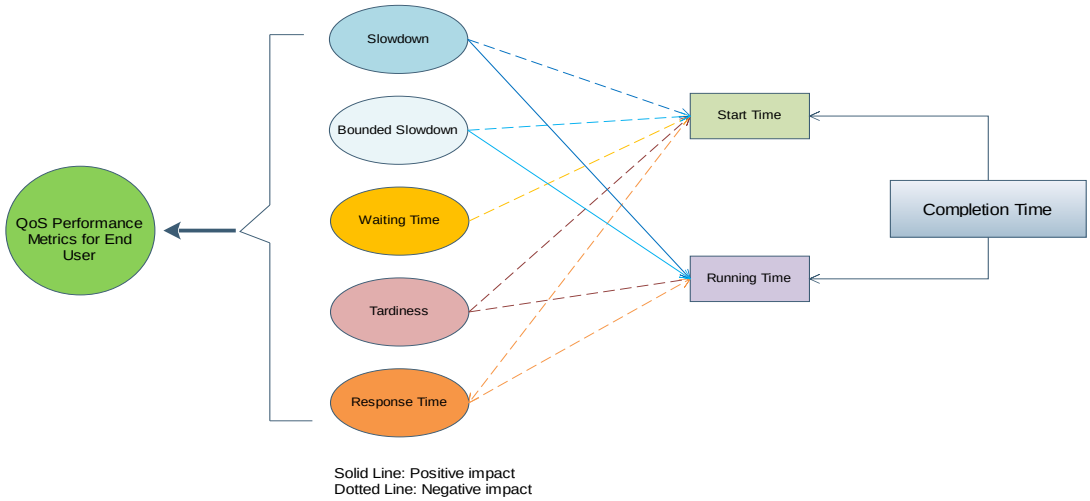


Figure 5.15. The influence of completion time on the performance metrics

In fact, the greater the runtime, the less will be the slowdown, explaining why short jobs are affected by the slowdown metric more than are long jobs. However, the effect of the start time is also significant. Even when the runtime value is small, if the start time value is high, this will affect the slowdown metric badly. Thus, in Figure 5.15 the line that links the slowdown/bounded slowdown metrics and the start time is dotted, while the line that links the slowdown/bounded slowdown metrics and runtime is solid.

The waiting time has no relationship with the runtime, although the greater the start time, the greater the waiting time will be. The submission time cannot be controlled, since the incoming time for the job is related to the users themselves. Therefore, in Figure 5.15 a dotted line links waiting time and start time and there is no line between the runtime and waiting time, since there is no relationship between them.

For tardiness, which represents the time difference between job completion and corresponding due date, it can be observed that to obtain a small tardiness value, it does not matter if the completion time and corresponding due date are great or small. The most important thing is that the difference between them is not large. However, the greater the completion time, the lower is the QoS with respect to the other metrics.

Since this research concerns the QoS for the end user, the CTS is designed to minimize tardiness through reducing completion time and bringing its value closer to the corresponding due date. The corresponding due date in this research is Pj, as explained in Equation 5.12. Therefore, based on the QoS criterion, the higher values of start and runtime are considered to have a negative effect.

This explains the dotted lines between the tardiness metric from one side and the start and runtimes from the other. Finally, the response time is linked by dotted lines to both

the start time and the runtime. This is evident, since the more the starting time the more response time, implicitly. In addition, more runtime increases the total response time.

5.4 Summary

This chapter presented the CTS, which improves the performance of SG. Reducing completion time can be achieved by reducing the start time value for the job, or minimizing the processing time by selecting a faster machine. In this research, the CTS covers both aspects. Selecting the best start time is achieved by avoiding the selection of the earliest gap if another machine's schedule ends sooner than the located earliest gap. Minimizing the processing time is based on selecting the faster machine to execute the job.

The extensive experiments have shown that the performance of CTS is better than SG alone in all performance metrics. Furthermore, the analysis of CTS performance has rationalized the variance in improvement ratio of the simulated performance metrics.

The following chapter presents Swift Gap with Completion Time Scheme (SG-CTS) and explains how CTS is integrated in the SG mechanism. It compares the performance of SG-CRT with additional scheduling mechanisms in order to evaluate the work fully.

CHAPTER SIX
SWIFT GAP WITH COMPLETION TIME SCHEME

This chapter explains how the Swift Gap mechanism (SG) is integrated with the developed Completion Time Scheme (CTS) and evaluates Swift Gap with Completion Time Scheme (SG-CTS) using three different real workloads. The evaluation considers mechanisms that implement the schedule-based and queue based approaches, and covers two techniques: backfilling and the priority rule.

The chapter is organized as follows. Integrating CTS with SG is presented in Section 6.1. Section 6.2 presents the evaluation of SG-CTS by comparing it with other mechanisms; this is followed by a comprehensive discussion. A general view of SG-CTS is introduced in Section 6.3. The chapter concludes with Section 6.4.

6.1 The Completion Time Scheme Integration with Swift Gap

The implementation of CTS in SG is applied in two stages: building the initial solution, and optimization.

In the first stage, the completion time is simply reduced through the best start time selection. The reason for ruling out the weighted machine, as explained in Section 4.3.3.1, is that only newly arrived jobs are handled. The priority here is to find a place for these new jobs to be backfilled. At this stage, the mechanism is unable to locate a faster machine; it can only detect the earliest gap that fits the job in the cluster. This is one of the limitations of the applied backfilling mechanism in this stage. The concept of CTS implementation in this stage is shown in Figure 6.1 followed by Algorithm 6.1. It is worth to be mentioned that Algorithm 5.1 explained before in Chapter Five, Section 5.1.1 will be applied in this stage before proceeding to Algorithm 6.1.

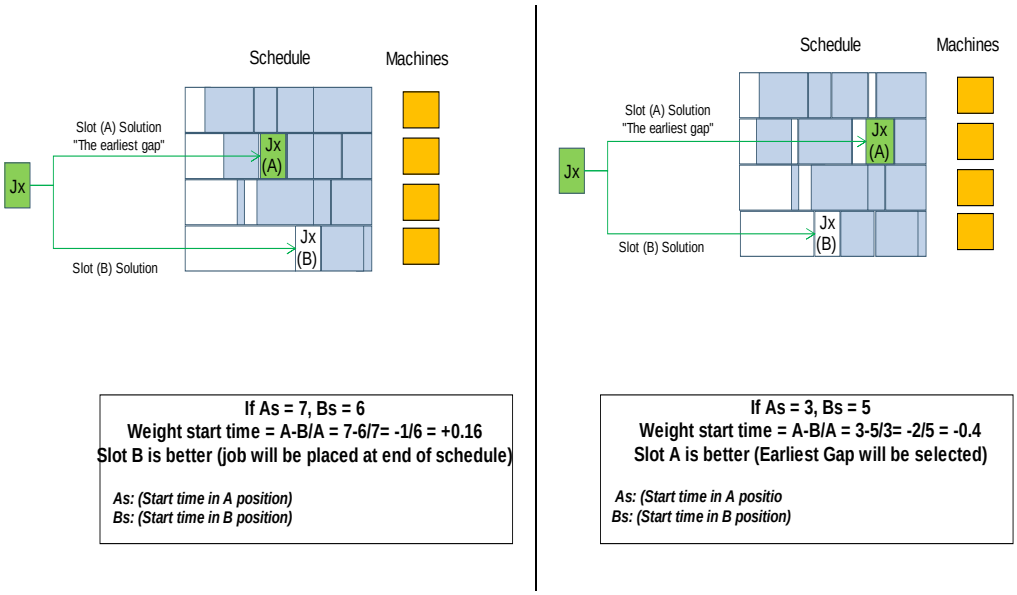


Figure 6.1. Implementing CTS in the building the initial solution stage

---

Algorithm 6.1 **CTS Integration in the Initial Solution**

---

1. get the start time for Slot (A);
2. get the start time for Slot (B) (the earliest gap selected);
3. weight $_{\text{start time}} = \frac{As-Bs}{As}$
4. total weight = weight $_{\text{start time}}$
5. if weight ≤ 0 then
6. reject the move and select Slot (A) as a solution;
7. else
8. accept the move and select Slot (B) as a solution;
9. end if

In the second stage, the implementation covers both rules (the selection of the best start time and weighted machine selection). The local search optimizing mechanism has a movement feature which enables it to move jobs both within the cluster and among other clusters [133]. Since the grid is heterogeneous, the impact of the weight machine implementation will be significant.

The implementation of best start time selection in the second stage guarantees that the faster machine will not necessarily be selected when the start time for a job in the selected machine is very late, compared to the slower machine. Thus, where there are two machines with different computational powers but the start time for the faster one is much later, a poor decision can be avoided, i.e. applying the best start time alongside with weighted machine selection strikes a balance for the job movement decision.

Figure 6.2 presents the case where only the weighted machine selection is implemented. Figure 6.2 illustrates the implementation of the CTS in the optimization stage, followed by Algorithm 6.2.

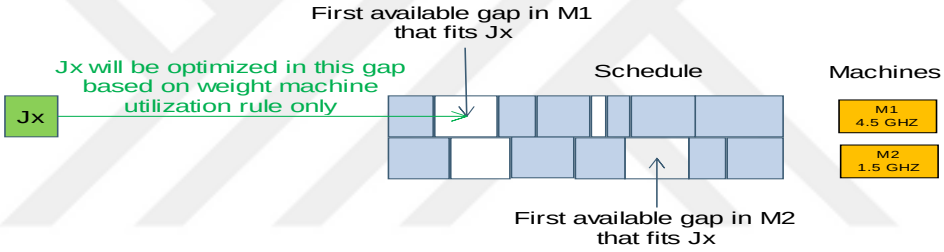


Figure 6.2. Unbalanced decision at the expense of the best start time selection

From Figure 6.1, based on the weight utilization selection, the proposed move for job Jx to the schedule of machine M2 will be rejected, because although M2's schedule contains the first gap that fits Jx, it is slower than M1. However, it can be observed that the best start time for the first available gap that fits Jx in M2's schedule is much better than M1's, so Jx has a better chance of starting and therefore finishing earlier, even though M1 is much faster than M2.

It is important to note that the proposed place for the job (Slot B) has been selected based on the earliest gap for comparison with Slot A, i.e. the mechanism in the next stage will select a machine provided that the gap in the machine's schedule fits the job

and it is the earliest one located. As mentioned above, the selected machine can belong to the cluster to which the optimized job belongs, or to another computing cluster.

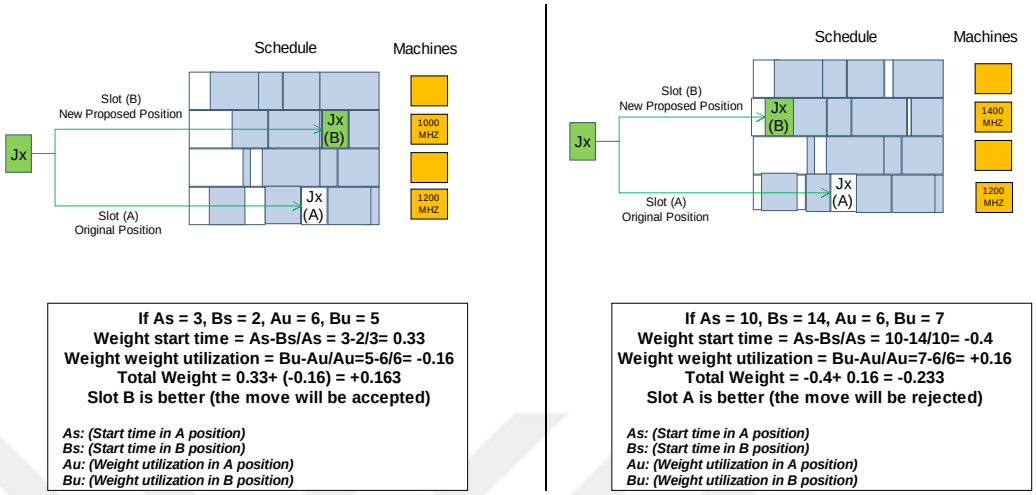


Figure 6.3. Implementing CTS in the optimization stage

If the total weight is equal to zero (weight $_{\text{start time}}$ + weight $_{\text{weight machine usage}}$ = 0), the total weight will be adjusted by adding a very small positive value, to make the total weight very close to zero (but not zero). The proposed move will therefore always be accepted when the total weight is equal to zero.

Algorithm 6.2 **CTS Integration in Optimization**

1. get the start time and the weight machine usage for Slot (A);
2. get the start time and the weight machine usage for Slot (B);
3. weight $_{\text{start time}} = \frac{As-Bs}{As}$
4. weight $_{\text{weight machine usage}} = \frac{Bu-Au}{Au}$
5. total weight = weight $_{\text{start time}}$ + weight $_{\text{weight machine usage}}$
6. if total weight ≤ 0 then
7. reject the move and select Slot (A) as a solution;
8. else
9. accept the move and select Slot (B) as a solution;
10. end if

6.2 The Evaluation of SG-CTS

This section provides an evaluation of SG-CTS against Conservative backfilling integrated with Gap Search (CONS-GS), Flexible Conservative backfilling (Flex-CONS) and Shortest Job First (SJF). CONS-GS employs the optimization technique, Flex-CONS is an extension of the original well-known mechanism CONS and SJF is one of the best classical mechanisms that implements a queue-based approach. Table 6.3 summarizes the features of the mechanisms simulated for the evaluation followed by graphical results where SG-CTS is the Left bar (L) and CON-GS is First Middle (M1), Flex -CONS is the Second Middle (M2) and SJF is the Right bar (R).

Table 6.1
Specifications of the evaluated mechanisms

Mechanism	Scheduling Approach	Backfilling Aspect	Priority	Optimization	Evaluation Statues
SG-CTS	Planning	Yes	No	Yes	To be evaluated in Chapter 6
SG	Planning	Yes	No	Yes	Evaluated in Chapter 4
CONS	Planning	Yes	No	No	Evaluated in Chapter 4
EASY	Queues	Yes	No	No	Evaluated in Chapter 4
CONS-GS	Planning	Yes	No	Yes	To be evaluated in Chapter 6
Flex-CONS	Planning	Yes	Partially	No	To be evaluated in Chapter 6
SJF	Queues	No	Yes	No	To be evaluated in Chapter 6

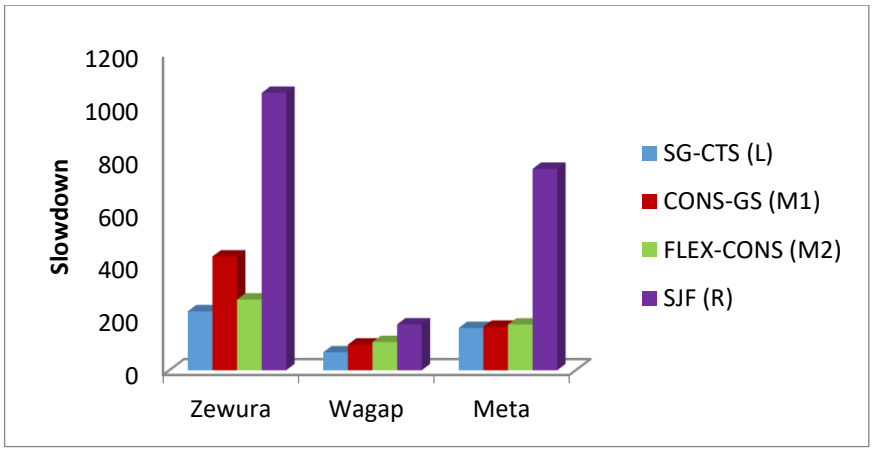


Figure 6.4. Slowdown for SG-CTS, CONS-GS, Flex-CONS and SJF

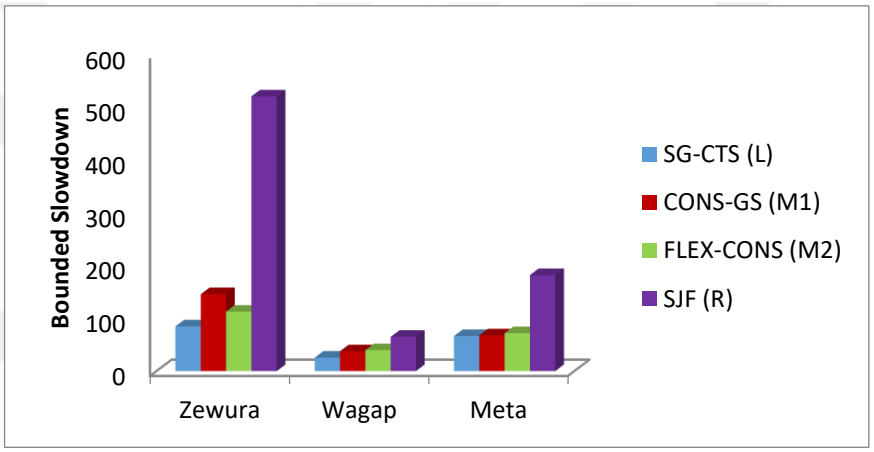


Figure 6.5. Bounded slowdown for SG-CTS, CONS-GS, Flex-CONS and SJF

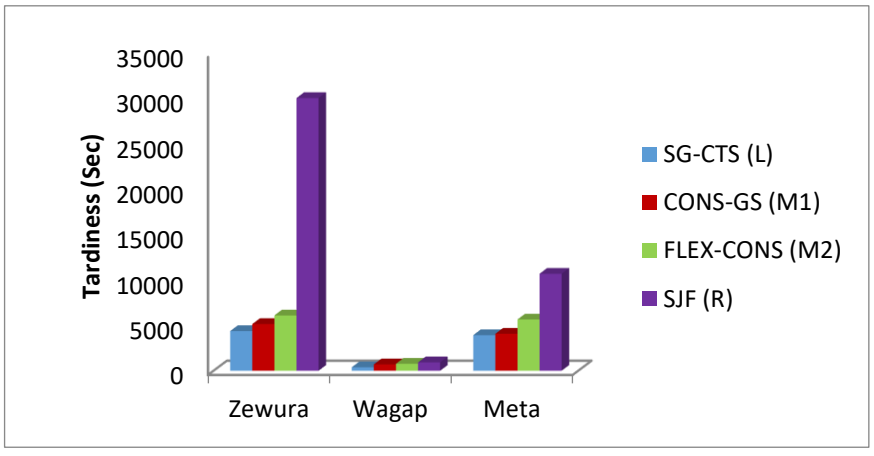


Figure 6.6. Tardiness for SG-CTS, CONS-GS, Flex-CONS and SJF

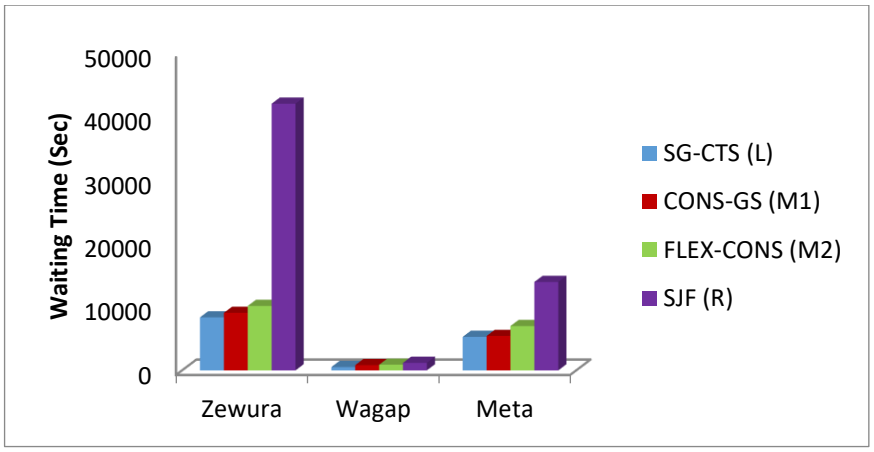


Figure 6.7. Waiting time for SG-CTS, CONS-GS, Flex-CONS and SJF

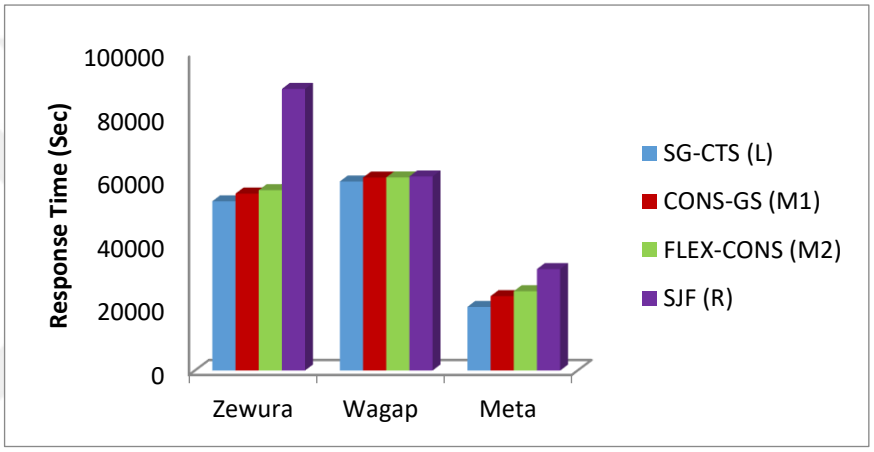


Figure 6.8. Response time for SG-CTS, CONS-GS, Flex-CONS and SJF

6.2.1 Results and Discussion

Table 6.2 shows the improvement ratio for the measured performance metrics in each simulated workload.

Table 6.2
SG-CTS vs CONS-GS, Flex CONS and SJF performance improvement ratio

Mechanism	Slowdown	Bounded Slowdown	Tardiness	Waiting Time	Response Time
CONS-GS (Zewura)	48.2%	41.8%	15%	7.9%	4.35%
CONS-GS (Wagap)	30.3%	32.2%	45.4%	37%	2%
CONS-GS (Meta)	3%	2%	3.8%	2.6%	14.5%
Flex-CONS (Zewura)	16.6%	24.7%	28.2%	17.7%	6.2%

Table 6.2 continued.

Flex-CONS (Wagap)	36.5%	35.7%	50.1%	41.3%	2.12%
Flex-CONS (Meta)	7.7%	7.5%	30.5%	24.4%	19.8%
SJF (Zewura)	78.7%	83.7%	+100%	80.1%	40%
SJF (Wagap)	62%	60.7%	58.6%	56%	2.6%
SJF (Meta)	79%	63.4%	63.3%	62%	37.2%

Table 6.2 demonstrates that SJF has the worst performance of all for each simulated workload. This is for two main reasons: SJF applies the queue-based approach besides fixed priority for the shortest jobs in the queue. It implicitly confirms the significance of the backfilling technique particularly for a heavily loaded system, exploiting the idle capacity in the processors better than classical fixed priority algorithms. However, SJF performs the best in the Wagap workload, which contains a higher number of short jobs than Zewura and Meta.

Flex-CONS performs better than the original CONS evaluated in Chapter Four. It works exactly like CONS except that it favours the job to be backfilled based on certain constraints. The performance of Flex-CONS deteriorates in the Wagap workload compared to other workloads, unlike the original CONS. This is because CONS favours the short jobs, as explained in Chapter Four, while Flex-CONS favours them less. This means that Flex-CONS recognizes other constraints besides the size of the job when backfilling, while the original CONS favours the jobs based only on their size.

CONS-GS outperforms all the other mechanisms except SG-CTS, as optimization is applied in this mechanism. However, SG-CTS outperforms CONS-GS because of CTS, as well as inheritance of the flaws related to the original CONS.

The slowdown and bounded slowdown metrics are related to the response time and the runtime. As previously explained, the greater the runtime that appears in the denominator, the more slowdown and bounded slowdown are minimized. Although CTS minimizes the runtime and that can produce a better slowdown, it also minimizes the start time that appears in the numerator. As discussed in Section 6.1, the start time sub-rule is applied twice, while the processing time sub-rule is applied in the second stage only. The result is that the start time is minimized more than the runtime.

In order to clarify this idea, the following numerical examples show three cases: first, when CTS is not applied; second, when only the runtime rule in CTS is applied; and third when the fully applied CTS is carried out in.

- Case 1: Let us suppose the start time (St) is 9, the submission time (Sb) is 5 and the runtime (Tr) is 12. The slowdown is $\frac{9-5+12}{12} = 1.33$.

- Case 2: Let us suppose the start time (St) is 9, the submission time (Sb) is 5 and the runtime (Tr) is 10.5. The slowdown is $\frac{9-5+10.5}{10.5} = 1.38$.

- Case 3: Let us suppose the start time (St) is 7, the submission time (Sb) is 5 and the runtime (Tr) is 10.5. The slowdown is $\frac{7-5+10.5}{10.5} = 1.19$.

The final case provides less slowdown, despite minimizing the runtime in the denominator. The question raised here is what values are proposed for the start time (St) and the runtime (Tr) in the third case. The values selected in this example might be not accurate; however, they are based on the revelation of the logic of the developed CTS and its implementation as well as the demonstrated evaluation results.

In fact, applying the processing time sub-rule will increase the slowdown, which is why this rule is represented in red and has a high value on the slowdown axis. The start time sub-rule improves the slowdown and even the response time, while the response time can be further improved by applying the processing time sub-rule to decrease the runtime. For this, the start time sub-rule is coloured blue and has an average value on the response time axis.

Both tardiness and waiting time metrics show improvement in SG-CTS over CONS-GS. Tardiness is the difference between the start time (St) and the corresponding due date, which is equal to $2 \times Tr$ as discussed Section 5.2.1, Equation (5.12). CTS reduces the start time (St) in stage one and stage two, and hence SG-CTS has the advantage of minimizing tardiness compared to CON-GS. The same is true for the waiting time, which is especially subject to the start time (St). Figure 6.9 shows the relation between slowdown and response time and how the CTS in the SG mechanism handles the conflicts between them to achieve the best level of QoS.

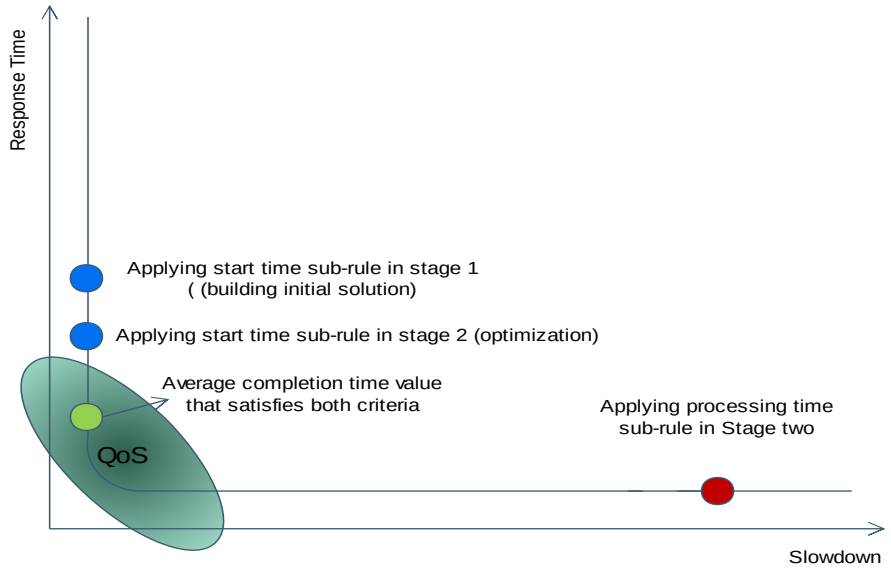


Figure 6.9. The relation between slowdown and response time

Response time in SG-CTS is improved over CON-GS, because the waiting time is reduced, as discussed above, as well as the runtime. The runtime is partially minimized by application of the processing time sub-rule. It can be observed from Table 6.4 that the response time improvement ratio is linked with the number of clusters in each workload. More clusters will reduce the response time.

This can be justified as follows: more clusters means the variation among the computational resources is more frequent, since the computational power can differ from one cluster to another, as listed in Table 6.1 (speed field). Thus, the processing time sub-rule has a higher chance of being applied more when the job is optimized from one cluster to another. Figure 6.11 compares the runtime in SG-CTS and the other mechanisms. SG-CTS is presented using the Left Bar (L) and Others are presented using Right bar (R).

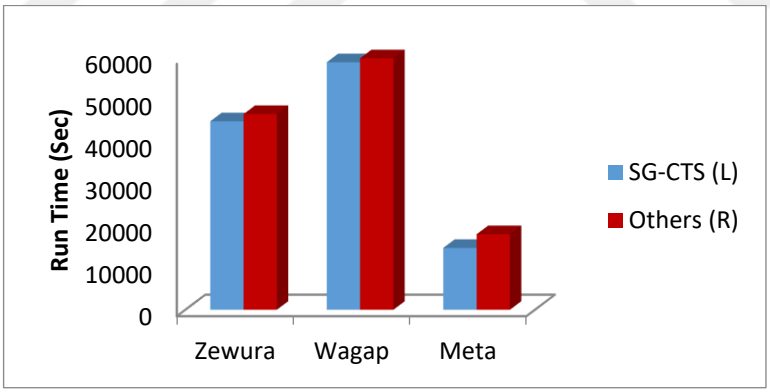


Figure 6.10. Run time for SG-CTS compared to other mechanisms

This slight improvement in the runtime depends on the number of clusters and the difference in speed scale among the clusters. Some clusters have the same speed; for instance, Meta has five clusters with a speed of 1, three with a speed of 2 and two with a speed of 3. All these factors together affect the minimized ratio of the runtime when the processing time sub-rule is applied.

6.2.2 SG-CRT and CONS-GS Real Time Comparison

The method used in this comparison was explained in Section 4.6.2. CONS-GS is the only one of the previously evaluated mechanisms selected, since it is the most competitive to SG-CRT.

Generally, the performance of SG-CTS is better than that of all the evaluated mechanisms in this study. It is difficult to determine in which workload it performs the best, since this is subject to the SG mechanism, which is hybridized, CTS, the workload and the measured metric itself [21]. However, it is clear that the improvement ratio in the Meta workload is the least (except for response time), while that in Zewura and Wagap is distributed almost equally among the performance metrics.

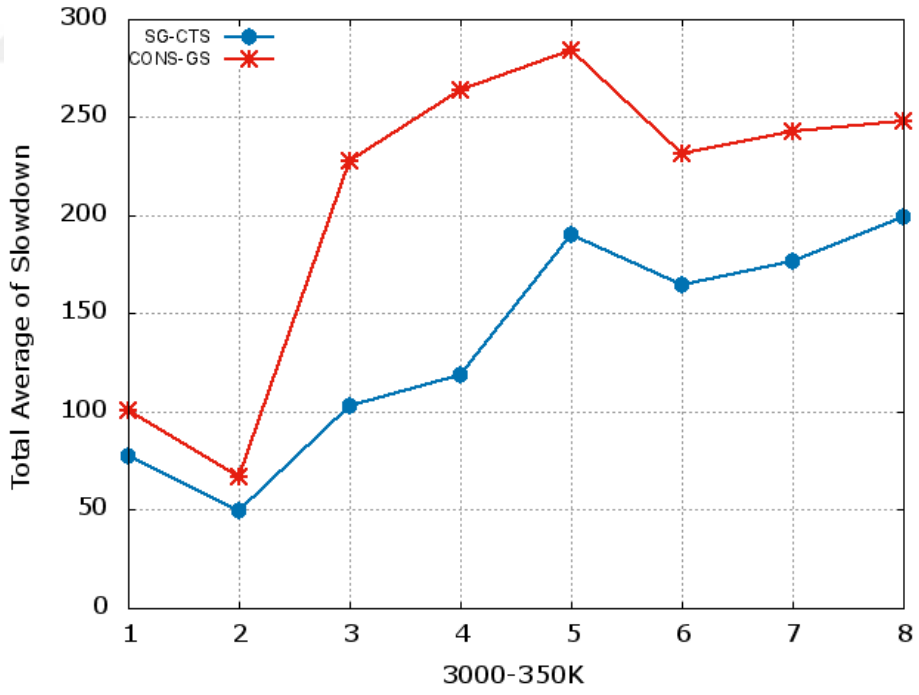


Figure 6.11. Total average of slowdown for SG-CTS and CONS-GS

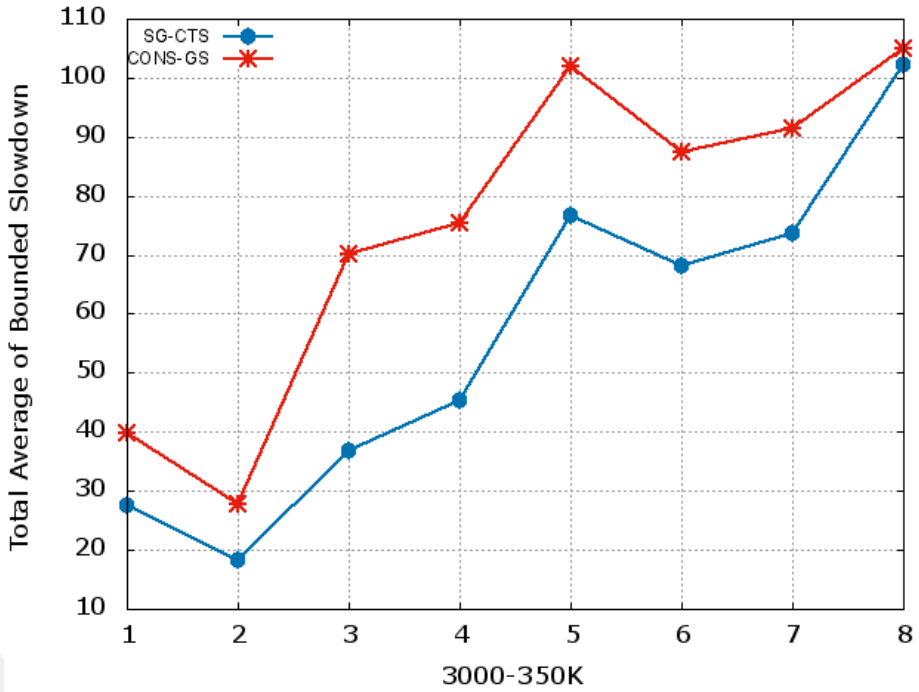


Figure 6.12. Total average of bounded slowdown for SG-CTS and CONS-GS

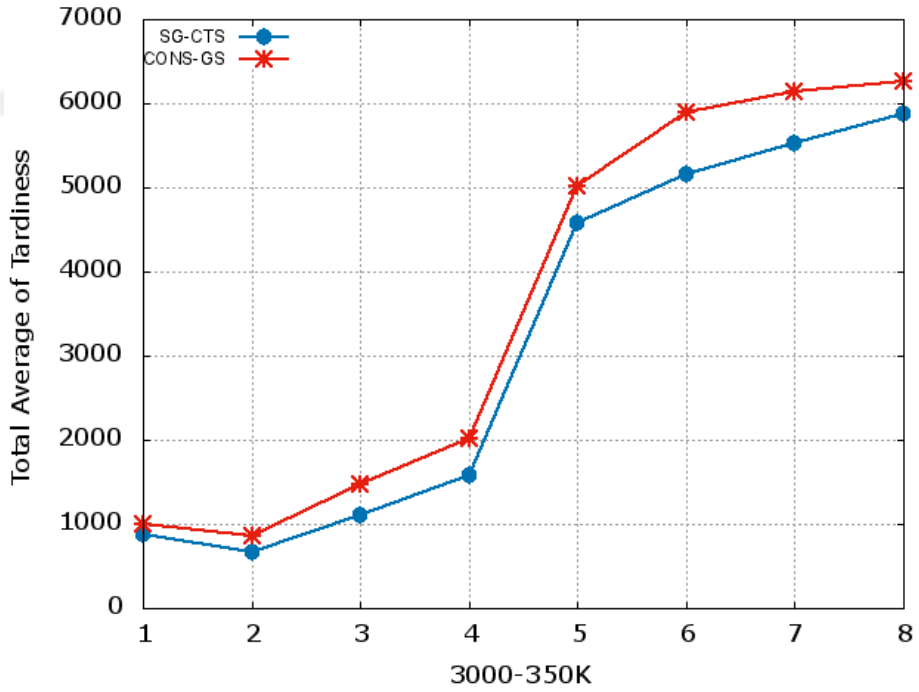


Figure 6.13. Total average of tardiness for SG-CTS and CONS-GS

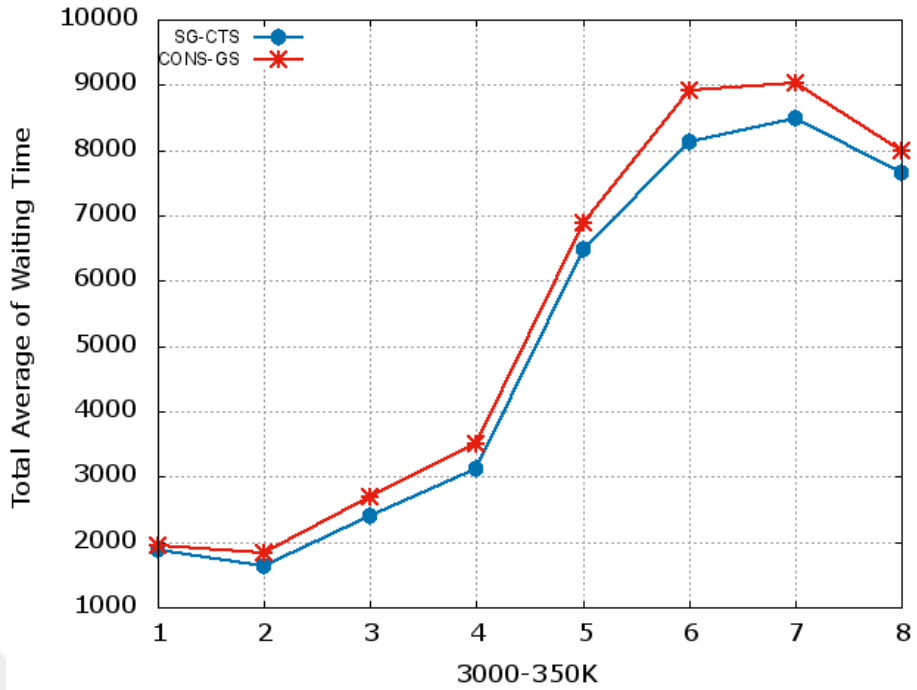


Figure 6.14. Total average of waiting time for SG-CTS and CONS-GS

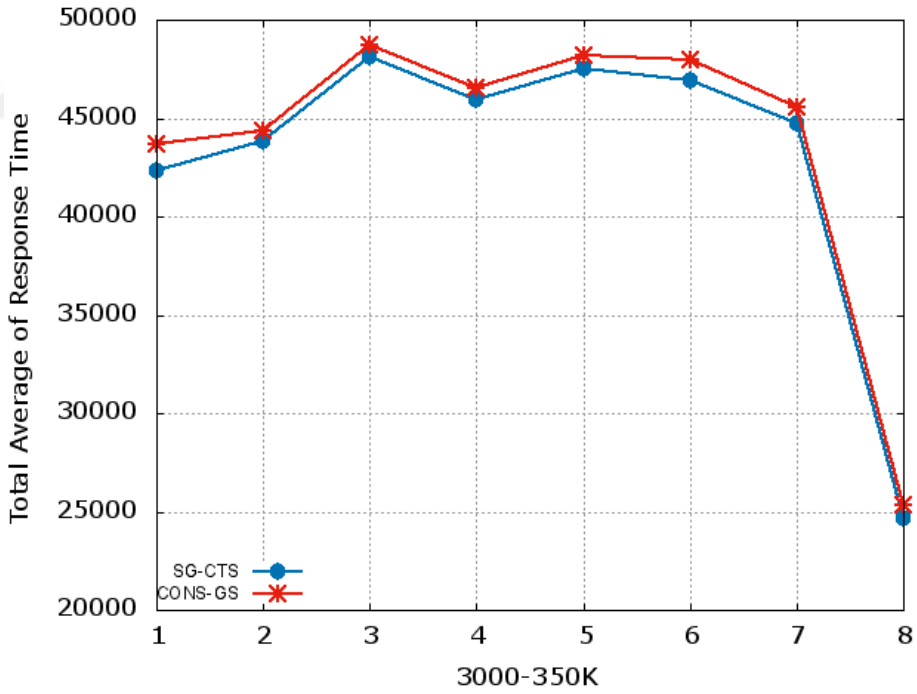


Figure 6.15. Total average of response time for SG-CTS and CONS-GS

Generally, the performance of SG-CTS is better than that of all the evaluated mechanisms in this study. It is difficult to determine in which workload it performs the

best, since this is subject to the SG mechanism, which is hybridized, CTS, the workload and the measured metric itself [21]. However, it is clear that the improvement ratio in the Meta workload is the least (except for response time), while that in Zewura and Wagap is distributed almost equally among the performance metrics.

Even with other mechanisms, such as CONS and EASY, the performance in the Meta workload shows a convergence between them. This might be rationalized as the workload containing many large Long Wide (LW) jobs. Long jobs in general are harder to backfill and the LW category has no specific trends in CONS and EASY. Hence, the improvement ratio in this workload is the least, compared to Zewura and Wagap.

6.3 Summary of SG-CTS

This section summarizes the proposed SG-CTS. First, it provides a bird view for the complete introduced work, and then it presents briefly how this mechanism operates.

6.3.1 General View of SG-CTS

When new jobs reach the system, Incremental Approach adds the new jobs without recreation the schedule from the scratch. Stage One starts by backfilling the jobs based on best start criterion. In the second stage, the optimization moves the jobs among the resources to further improve the initial solution. Moving job criteria in this stage is based on the best start time and the weight machine. When the stopping conditions in the Stage Two are true, the best solution will be executed. Finally, if a sudden event such as job termination, machine failure or restart occurs, Anytime Approach aborts the optimization in order to enable the first stage to handle the new events. Incremental

Approach and Anytime Approach are enabled by means of Planning Approach. Figure 6.16 presents the general view of SG-CTS.

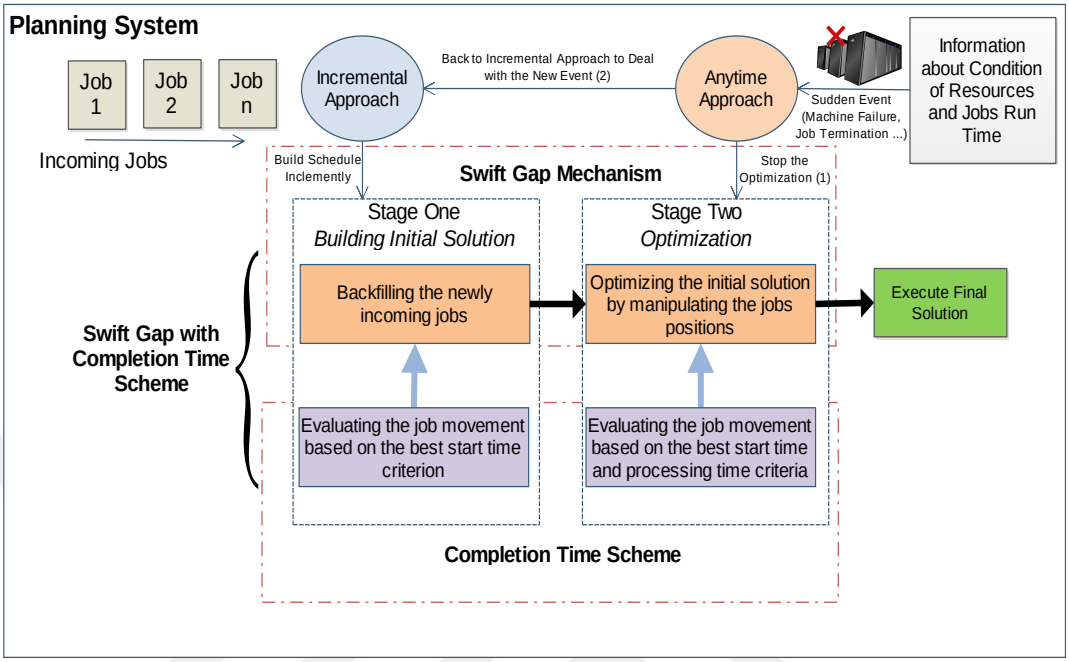


Figure 6.16. General view of SG-CTS

6.3.2 SG-CTS Working Steps

SG-CTS runs starting from Incremental Approach. The first step is to find a gap that fits the arrival job. The selected gab is based on the earliest gap. Then, Algorithm 5.1 introduced in Section 5.1.1 checks if the earliest gap represents the best start time for the job or not. If yes, the job will stay in the current place. Otherwise, Algorithm 5.2 introduced in Section 5.1.1 will be applied.

The optimization based on Tabu Search (TS) selects random job to optimize as discussed in Section 4.3.3.2. The optimization criteria are based on Algorithm 6.1 and 6.2 as introduced in Section 6.1. If one of the stopping conditions are achieved, the final solution will be executed. Anytime Approach interrupts the optimization if a sudden event has occurred. Figure 6.17 illustrates how SG-CTS operates.

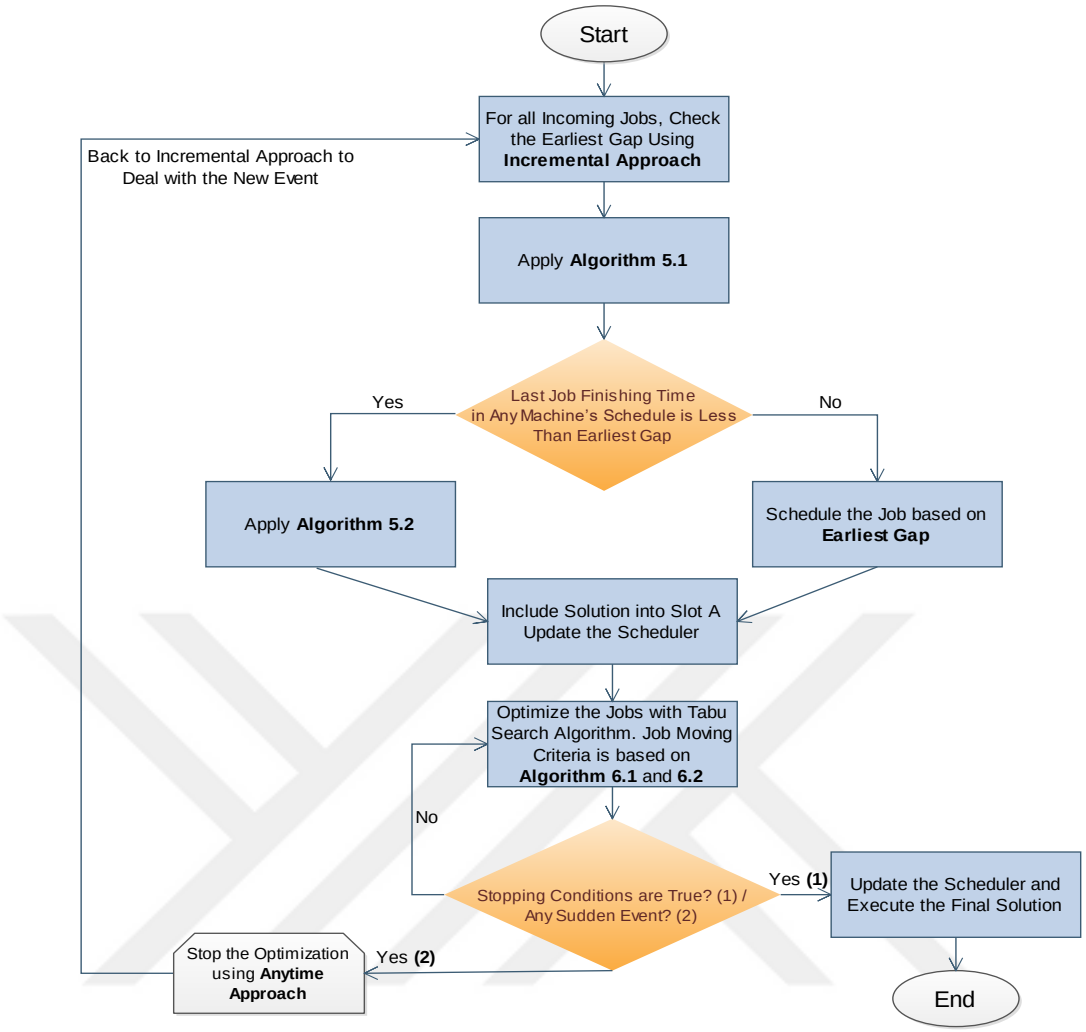


Figure 6.17. SG-CTS working steps

6.4 Summary

This chapter described the results of integrating CTS in SG, with full justification for which part of this scheme has to be applied in each stage. It introduced an extensive evaluation of SG-CTS using three real workloads, each specified differently, by number of clusters, speed of clusters and the number of machines in each cluster. The simulation covered a large number of jobs, in order to imitate a real grid computing environment. SG-CTS was evaluated against three scheduling mechanisms, CONS-GS, Flex-CONS and SJF. The measured performance metrics concern the QoS for the end user. The results demonstrate that SG-CTS outperforms all the simulated mechanisms for all the measured performance metrics.

CHAPTER SEVEN
CONCLUSION AND FUTURE WORK

This research aimed to provide a new scheduling mechanism, called Swift Gap (SG), for computational grid environment. The performance of the proposed mechanism has been evaluated extensively through simulation. A conclusion to the study is provided in this chapter. Section 7.1 summarizes the research and its findings. The contributions of the study are highlighted in Section 7.2, while Section 7.3 addresses its limitations. Finally, Section 7.4 provides some suggestions for further work.

7.1 Research Summary

The dynamic nature of grid computing motivated the researcher to propose mechanisms to handle the scheduling problem in this environment. Although a plethora of mechanisms is described in the literature, there is still an absence of scheduling mechanisms that can deal with real grid environments when millions of incoming jobs reach the system.

As explained in Chapter One, the classical scheduling mechanisms are simple and efficient for small grid systems. However, they perform poorly when the number of jobs becomes huge, especially in dynamic grid systems that run for a long time. Later, the backfilling approach emerged, succeeding in improving the efficiency of the whole system by fully utilizing the computational resources. However, backfilling mechanisms still suffer from several performance issues such as long delay, long waiting time for the larger jobs and a low Quality of Service (QoS) level. The shorter jobs suffer from long delay due to the aggressive approach applied in the backfilling, while the larger jobs experience long waiting times due to the restrictive approach that operated in backfilling.

Chapter Two compared the planned and queue scheduling approaches. According to the literature, the former has several unique features, which motivated this study to implement this approach. In addition, well-known mechanisms were critically reviewed, providing a deeper understanding of performance issues in scheduling mechanisms. This inspired the scheduling mechanisms to implement optimization techniques to enhance the performance. Therefore, Tabu Search (TS) features were studied and introduced.

To fulfil the objectives of this research, Design Research Methodology (DRM) was adopted as a framework and guideline. The simulation process proposed by Jain [142] and considered criteria [8] were adopted to successfully define the research objectives.

First, the SG mechanism was designed, based on the integration of two mechanisms. The first provided an initial solution by backfilling newly arrived jobs, while the second applied an optimization mechanism to improve the initial solution. The hybridization was based on the idea that the output of the first mechanism would be introduced as input to the second mechanism.

Next, Completion Time Scheme (CTS) was developed to improve the performance of SG. CTS reduces the start and processing times using a weighting function. The integration was applied twice; in the first stage, only the start time sub-rule was applied, and in the second both the start time and processing time were applied. The developed CTS improved performance by providing the best start time for the job and utilizing the fastest resources.

Evaluation of the SG scheduling mechanism confidently confirmed that the aim and objectives of this research have been well and fully accomplished.

7.2 Research Contributions

The main contribution of this research is the design of a scheduling mechanism for computational grid environments, to reduce the delay for jobs sent by end users. A new mechanism employing multi-level scheduling was proposed in order to improve the scheduling efficiency. The CTS mechanism improved SG's performance. The detailed contributions of this research are as follows:

i. The development of a performance model for the SG mechanism with CTS over grid computing can serve as an indicator for any intended improvement and enhancement related to the QoS criteria for the end user in this dynamic environment.

ii. The minimizing of the delay metrics in scheduling for High Performance Computing applications (HPC) in grid computing was achieved by introducing a new scheduling mechanism, SG

    a. Applying a planned scheduling approach in order to schedule the jobs in advance and make reservations for them.

    b. Applying multi-level scheduling to enable applications with many short-running tasks to execute efficiently on grids.

    c. Integrating the Best Gap mechanism with TS, resulting in a loosely coupled hybridization scheme. The applied approach in (a) enables implementing this type of hybridization.

iii. Further improvement of the QoS for the end user by introducing CTS.

    a. Reducing the start time using the weight function.

b. Reducing the start time using the weight function.

c. Integrating CTS with the SG mechanism. The first stage applies only the start time sub-rule since the priority is to find a right place for the job first and Best Gap mechanism cannot detect the faster machine, while the second implementation covers the start time and the processing time.

iv. The influence of completion time on the performance metrics was clarified by illustrating the relationship between the QoS for the end user in grid computing environments and the simulated performance metrics.

7.3 Research Limitations

Although this study was carried out under careful selection of a set of supporting methods and guidelines, it is limited to particular and specific usage. First, the running jobs referred to parallel and sequential jobs only, while experiments for workflow applications were not conducted since jobs sent by HPC applications belong to parallel and sequential type of jobs. The maximum number of simulated jobs was 350,000, a small figure compared with real grid environments.

Further massive, recent and recommended workloads are hard to obtain. The experiments in this research were concerned only with CPUs, and other types of resource were not considered in this study. Finally, the evaluation was carried out using only simulation, since implementing the proposed mechanism in a real grid would be a complicated process.

7.4 Future Work

The proposed mechanisms in this research have improved the scheduling performance and QoS in grid computing environments,. Some possible directions for future work, including rectifying the limitations outlined above, are as follows.

a) Extending the application type: the simulated jobs in Chapters Four, Five and Six are HPC applications. However, a more precise verdict on the proposed mechanisms could be given if other types of application are considered.

b) Covering network aspects: HPC applications are applied in small area networks, but if other application types are considered, network aspects should be investigated. When an application runs over a wide area network, some issues related to link failure and time out when sending requests for resources have to be addressed.

c) Conducting a failure scenario: in HPC applications, the resources are dedicated to the requests sent by users, hence failure scenarios are uncommon. However, for applications that run over distributed geographical zones in grids, failure scenarios are quite possible. The scheduler and the resource may fail to communicate due to bandwidth or connectivity issues, or the resource may sign out/in or reboot suddenly for many reasons. In this case, the applied scheduling mechanism will face a new challenge in order to re-assign the job to other available resources and maintain the QoS at an acceptable level.

d) Simulating other resource types: although the CPU is the most important resource type in the computational grid, other types such as RAM and Hard Drive Disks (HDD) might be considered. When the CPU is executing the jobs, all the results generated from processing the complex calculations are saved in RAM, then HDD reads the

stored data in RAM in order to enable the running application to reach the outcome data. Hence, efficient CPU with insufficient RAM will make the running application slower. Therefore, the scheduling policy will guarantee smoother running of the application if it schedules multiple types of resource and coordinates among them according to user need.

Evaluating the scheduling mechanism in a real environment: although the mechanisms were extensively evaluated using real workloads, yet extending the evaluation to a real grid computing environment would be very interesting. Furthermore, extending the evaluation to a real environment may reveal new challenging issues. This would certainly be a possible way to extend the life of this research.

REFERENCES

[1] J. Joseph, M. Ernest, and C. Fellenstein, "Evolution of grid computing architecture and grid adoption models," *IBM Systems Journal*, vol. 43, pp. 624-645, 2004.

[2] D. L. Massimo Cafaro, Sandro Fiore, Giovanni Aloisio, Robert Van Engelen, "The GSI plug-in for gSOAP: building cross-grid interoperable secure grid services," in *Parallel Processing and Applied Mathematics*, 2008, pp. 894-901.

[3] M. B. Qureshi, M. M. Dehnavi, N. Min-Allah, M. S. Qureshi, H. Hussain, I. Rentifis, *et al.*, "Survey on Grid Resource Allocation Mechanisms," *Journal of Grid Computing*, pp. 1-43, 2014.

[4] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," in *Proceedings on High Performance Distributed Computing, Symposium 10th IEEE International* pp. 181-194, 2001.

[5] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International journal of high performance computing applications*, vol. 15, pp. 200-222, 2001.

[6] I. Foster and C. Kesselman, "Computational Grids," *Cern European Organization for Nuclear Research-Reports-Cern*, pp. 87-114, 1998.

[7] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan, "Analyzing market-based resource allocation strategies for the computational grid," *International Journal of High Performance Computing Applications*, vol. 15, pp. 258-281, 2001.

[8] E. Frachtenberg and D. G. Feitelson, "Pitfalls in parallel job scheduling evaluation," in *Workshop on Job Scheduling Strategies for Parallel Processing*, 2005, pp. 257-282.

[9] H. Casanova, "Distributed computing research issues in grid computing," *ACM SIGAct News*, vol. 33, pp. 50-70, 2002.

[10] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed resource management for high throughput computing," in *The Seventh International Symposium on High Performance Distributed Computing*, 1998, pp. 140-146, .

[11] D. Klusacek and H. Rudova, "Improving QoS in computational Grids through schedule-based approach," in *Scheduling and Planning Applications Workshop at the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008), Sydney, Australia*, 2008.

[12] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE'08*, 2008, pp. 1-10.

[13] M. Hovestadt, O. Kao, A. Keller, and A. Streit, "Scheduling in HPC resource management systems: Queuing vs. planning," in *Workshop on Job Scheduling Strategies for Parallel Processing*, 2003, pp. 1-20.

[14] A. W. M. Alem and D. G. Feitelson, "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, pp. 529-543, 2001.

[15] C. B. Lee, "On the User-Scheduler Relationship in High-Performance Computing," Ph.D Dissertation, University of California, Departmnet of Computer Science, USA, SAN DIEGO, 2009.

[16] O. Dakkak, S. A. Nor, and S. Arif, "Scheduling through Backfilling Technique for HPC Applications in Grid Computing Environment," in *IEEE Conference on Open Systems (ICOS)*, 2016, pp. 30-35.
[17] O. Dakkak, S. A. Nor, and S. Arif, "A Critical Review on Resource Allocation Mechanisms in Grid Computing," presented at the The 4th International Conference on Internet Applications, Protocols and Services (NETAPPS2015) Cyberjaya, Malaysia 2015.
[18] O. Dakkak, S. Arif, and S. A. Nor, "A Critical Analysis of Simulators in Grid," *Jurnal Teknologi, UTM*, vol. 77, pp. 111-117, 2015.
[19] Z. R. M. Azmi, K. A. Bakar, M. S. Shamsir, W. N. W. Manan, and A. H. Abdullah, "Scheduling Grid Jobs Using Priority Rule Algorithms and Gap Filling Techniques," *International Journal of Advanced Science and Technology*, vol. 37, pp. 61-76, 2011.
[20] D. Lifka, "The anl/ibm sp scheduling system," in *Job scheduling strategies for parallel processing*, pp. 295-303, 1995.
[21] A. W. Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, pp. 529-543, 2001.
[22] D. T. Klusáček, Šimon, "On the Challenges in the Design of Efficient Job Scheduling Policies for Production HPC and Grid Environments," *Nova Science Publishing*, pp. 71-98, 2015.
[23] J. Cao, K. Li, and I. Stojmenovic, "Optimal power allocation and load distribution for multiple heterogeneous multicore server processors across clouds and data centers," *IEEE Transactions on Computers*, vol. 63, pp. 45-58, 2014.
[24] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy, "A distributed resource management architecture that supports advance reservations and co-allocation," in *Workshop on Quality of Service, 1999. IWQoS'99. 1999 Seventh International* 1999, pp. 27-36.
[25] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde, "Falkon: a Fast and Light-weight tasK executiON framework," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, 2007, p. 43.
[26] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International journal of high performance computing applications*, vol. 15.3 pp. 200-222, 2001 2003.
[27] F. Baker, "Requirements for IP version 4 routers," Cisco Systems, Santa Barbara, California 93111, USA, 1995.
[28] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," in *IFIP international conference on network and parallel computing*, pp. 2-13, 2005.
[29] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *International Journal of High Performance Computing Applications*, vol. 11, pp. 115-128, 1997.
[30] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, *et al.*, "Data management and transfer in high-performance computational grid environments," *Parallel Computing*, vol. 28, pp. 749-771, 2002.
[31] J. M. Schopf, L. Pearlman, N. Miller, C. Kesselman, I. Foster, M. D'Arcy, *et al.*, "Monitoring the grid with the Globus Toolkit MDS4," in *Journal of Physics: Conference Series*, 2006, p. 521.
[32] F. Berman, "High-performance schedulers," *The grid: blueprint for a new computing infrastructure*, vol. 67, pp. 279-309, 1999.

[33] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, "Application-level scheduling on distributed heterogeneous networks," in *Conference on Supercomputing, 1996. Proceedings of the 1996 ACM/IEEE* 1996, pp. 39-39.
[34] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke, "The community authorization service: Status and future," *arXiv preprint cs/0306082*, 2003.
[35] S. N. Pardeshi, C. Patil, and S. Dhumale, "Grid Computing Architecture and Benefits," *International Journal of Scientific and Research Publications*, p. 781.
[36] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour, "Evaluation of job-scheduling strategies for grid computing," in *Grid Computing—GRID 2000*, ed: Springer, 2000, pp. 191-202.
[37] F. Dong and S. G. Akl, "Scheduling algorithms for grid computing: State of the art and open problems," Technical report, 2006.
[38] H. J. Siegel, H. G. Dietz, and J. K. Antonio, "Software support for heterogeneous computing," *ACM Computing Surveys (CSUR)*, vol. 28, pp. 237-239, 1996.
[39] A. A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C.-L. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer*, vol. 26, pp. 18-27, 1993.
[40] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, *et al.*, "A Resource Management Architecture for Metacomputing Systems," in *Job Scheduling Strategies for Parallel Processing*, 1998, pp. 62-82.
[41] L. Ismail, "Dynamic resource allocation mechanisms for grid computing environment," in *Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities*, 2007, pp. 1-5.
[42] T. Wu*, N. Ye, and D. Zhang, "Comparison of distributed methods for resource allocation," *International Journal of Production Research*, vol. 43, pp. 515-536, 2005.
[43] A. C. Enterprises. *Inc. TORQUE Admininstrator Guide, version 3.0. 3. February 2012*. Available: http://www.clusterresources.com/products/mwm/docs/
[44] D. Jackson, Q. Snell, and M. Clement, "Core algorithms of the Maui scheduler," in *Workshop on Job Scheduling Strategies for Parallel Processing*, 2001, pp. 87-102.
[45] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke, "SNAP: A Protocol for Negotiation of Service Level Agreements and Coordinated Resource Management in Distributed Systems, submission to Job Scheduling Strategies for Parallel Processing Conference (JSSPP)," *SSPP), April*, vol. 30, 2002.
[46] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, "Theory and practice in parallel job scheduling," in *Workshop on Job Scheduling Strategies for Parallel Processing*, 1997, pp. 1-34.
[47] L. P. Huse and O. W. Saastad, "The Network Agnostic MPI–Scali MPI Connect," in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, 2003, pp. 294-301.
[48] M. DeBergalis, P. F. Corbett, S. Kleiman, A. Lent, D. Noveck, T. Talpey, *et al.*, "The Direct Access File System," in *FAST*, 2003, pp. 175-188.
[49] W. Smith, V. Taylor, and I. Foster, "Using run-time predictions to estimate queue wait times and improve scheduler performance," in *Workshop on Job Scheduling Strategies for Parallel Processing*, 1999, pp. 202-219.

[50] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a new computing infrastructure*: Elsevier, 2003.
[51] J. MacLaren, V. Sander, and W. Ziegler, "Advance reservations: State of the Art," in *Global Grid Forum*, 2003.
[52] O.-W. D. Snelling, "OGSA Fundamental Services: Requirements for Commercial GRID Systems," 2002.
[53] D. C. Verma, "Supporting service level agreements on IP networks," *Macmillan Technical Publishing*, pp. 1-13, 1999.
[54] A. Sahai, A. Durante, and V. Machiraju, "Towards automated SLA management for web services," *Hewlett-Packard Research Report HPL-2001-310 (R. 1)*, 2002.
[55] A. Sahai, V. Machiraju, M. Sayal, A. Van Moorsel, and F. Casati, "Automated SLA monitoring for web services," in *International Workshop on Distributed Systems: Operations and Management*, 2002, pp. 28-41.
[56] K. Somasundaram and S. Radhakrishnan, "Task resource allocation in grid using swift scheduler," *International Journal of Computers, Communications & Control*, vol. 42, pp. 158-166, 2009.
[57] J. Bansal, S. Rani, and P. Singh, "A Novel Heuristic for Scheduling of Independent jobs on Grid Resources," *International Journal of Engineering and Technology (IJET)*, vol. 7, pp. 2122-2129, 2016.
[58] F. Xhafa, L. Barolli, and A. Durresi, "Batch mode scheduling in grid systems," *International Journal of Web and Grid Services*, vol. 3, pp. 19-37, 2007.
[59] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and computation: practice and experience*, vol. 14, pp. 1175-1220, 2002.
[60] D. s. support. (2012). *Flex-CONS in Alea Simulator*. Available: http://www.fi.muni.cz/~xklusac/index.php
[61] D. s. support. (2015). *CONS-GS in Alea Simulator*. Available: http://www.fi.muni.cz/~xklusac/alea/index.html
[62] F. Glover, "Tabu search-part I," *ORSA Journal on computing*, vol. 1, pp. 190-206, 1989.
[63] F. Glover and M. Laguna, "Tabu Search∗," ed New York: Springer, 2013.
[64] F. Glover, M. Laguna, and R. Marti, "Principles of tabu search," *Approximation Algorithms and Metaheuristics*, vol. 23, pp. 1-12, 2007.
[65] F. Glover, "Tabu search—part II," *ORSA Journal on computing*, vol. 2, pp. 4-32, 1990.
[66] F. Glover, "Tabu search: A tutorial," *Interfaces*, vol. 20, pp. 74-94, 1990.
[67] S. Csiszar, "Optimization algorithms (survey and analysis)," in *2007 International Symposium on Logistics and Industrial Informatics*, 2007, pp. 185-188.
[68] O. Steinmann, A. Strohmaier, and T. Stützlel, "Tabu search vs. random walk," in *Annual Conference on Artificial Intelligence*, 1997, pp. 337-348.
[69] L. T. Blessing and A. Chakrabarti, *DRM, a Design Research Methodology*: Springer Science & Business Media, 2009.
[70] P. Offermann, O. Levina, M. Schönherr, and U. Bub, "Outline of a design science research process," in *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, 2009, p. 7.
[71] R. Sagban, K. R. K. Mahamud, and M. S. A. Bakar, "Reactive memory model for ant colony optimization and its application to TSP," in *Conference on*

Control System, Computing and Engineering (ICCSCE) , 2014 IEEE International, 2014, pp. 310-315.
[72] K. R. Ku-Mahamud, "Hybrid ant colony system and flower pollination algorithms for global optimization," in *Conference on IT in Asia (CITA), 2015 9th International*, 2015, pp. 1-9.
[73] M. M. Alobaedy and K. R. Ku-Mahamud, "Scheduling jobs in computational grid using hybrid ACS and GA approach," in *Computing, Communications and IT Applications Conference (ComComAp), 2014 IEEE*, 2014, pp. 223-228.
[74] A. Habbal, "TCP SINTOK: Transmission Control Protocol with Delay-based Loss Detection and Contention Avoidance Mechanisms For Mobile Ad hoc Networks," Ph.D Dissertation, School of Computing , Universiti Utara Malaysia, Malaysia 2014.
[75] O. Dakkak, S. A. Nor, and S. Arif, "A Critical Review on Resource Allocation Mechanisms in Grid Computing," presented at the Internet Applications, Protocols and Services (NETAPPS2015), 2015.
[76] M. Guizani, A. Rayes, B. Khan, and A. Al-Fuqaha, *Network modeling and simulation: a practical perspective*: John Wiley & Sons, 2010.
[77] O. Balci, "Verification validation and accreditation of simulation models," in *Proceedings of the 29th conference on Winter simulation*, 1997, pp. 135-141.
[78] R. G. Sargent, "Verification and validation of simulation models," in *Proceedings of the 37th conference on Winter simulation*, 2005, pp. 130-143.
[79] S. Microsystems, "NetBeans Integrated Development Environment (IDE)," version 6.1 ed, pp. Software Development Platform, 2014.
[80] "QJ-PRO: A Code Analyzer for Java," ed: open source project, pp. static analysis of Java source, 2004.
[81] E. Foundation, "Eclipse Integrated Development Environment (IDE)," ed.
[82] O. Balci, "Verification, validation, and testing," *Handbook of simulation*, pp. 335-393, 1998.
[83] R. B. Whitner and O. Balci, "Guidelines for selecting and using simulation model verification techniques," in *Proceedings of the 21st conference on Winter simulation*, 1989, pp. 559-568.
[84] L. G. Birta and F. N. Özmizrak, "A knowledge-based approach for the validation of simulation models: the foundation," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 6, pp. 76-98, 1996.
[85] R. S. Pressman, *Software engineering: a practitioner's approach*: Palgrave Macmillan, 2005.
[86] G. J. Myers, C. Sandler, and T. Badgett, *The art of software testing*: John Wiley & Sons, 2011.
[87] A. HABBAL, "TCP Sintok: Transmission Control Protocol with Delay-Based Loss Detection and Contention Avoidance Mechanisms for Mbile AD-HOC Networks," Ph.D Dissertation, School of Computing, Universiti Utara Malaysia, Malaysia, 2014.
[88] B. Beizer, *Software testing techniques*: Dreamtech Press, 2003.
[89] R. L. Van Horn, "Validation of simulation results," *Management Science*, vol. 17, pp. 247-258, 1971.
[90] C. F. Hermann, "Validation problems in games and simulations with special reference to models of international politics," *Behavioral science*, vol. 12, pp. 216-231, 1967.
[91] I. Sommerville, "Software Engineering. International computer science series," *ed: Addison Wesley*, 2004.

[92] K. J. Cohen and R. M. Cyert, "Computer models in dynamic economics," *The Quarterly Journal of Economics*, pp. 112-127, 1961.
[93] J. W. Forrester, "Industrial dynamics," *Journal of the Operational Research Society*, vol. 48, pp. 1037-1041, 1997.
[94] D. Miller, "Validation of computer simulations in the social sciences," in *Proceedings of the Sixth Annual Conference on Modeling and Simulation*, 1975, pp. 743-746.
[95] R. D. Wright, "Validating dynamic models: An evaluation of tests of predictive power," in *Proceedings of the 1972 Summer Computer Simulation Conference*, 1972, pp. 1286-1296.
[96] F. Xhafa and A. Abraham, "Computational models and heuristic methods for Grid scheduling problems," *Future generation computer systems*, vol. 26, pp. 608-621, 2010.
[97] J. Gomoluch and M. Schroeder, "Market-based Resource Allocation for Grid Computing: A Model and Simulation," in *Middleware Workshops*, 2003, pp. 211-218.
[98] P. Fibich, L. Matyska, and H. Rudová, "Model of grid scheduling problem," *Exploring Planning and Scheduling for Web Services, Grid and Autonomic Computing*, pp. 17-24, 2005.
[99] C. Ernemann, V. Hamscher, and R. Yahyapour, "Benefits of global grid computing for job scheduling," in *Workshop on Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International* 2004, pp. 374-379.
[100] S. Hassan, "Simulation-based Performance Evaluation of TCP-friendly Protocols for Supporting Multimedia Applications in the Internet," *School of Computing*, 2002.
[101] J. Mo, "Performance modeling of communication networks with markov chains," *Synthesis Lectures on Data Management*, vol. 3, pp. 1-90, 2010.
[102] J.-Y. Le Boudec, *Performance evaluation of computer and communication systems*: EPFL Press, 2010.
[103] R. Jain, *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*: John Wiley & Sons, 1990.
[104] L. F. Perrone and Y. Yuan, "Modeling and simulation best practices for wireless ad hoc networks," in *Simulation Conference, 2003. Proceedings of the 2003 Winter*, 2003, pp. 685-693.
[105] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya, "A toolkit for modelling and simulating data Grids: an extension to GridSim," *Concurrency and Computation: Practice and Experience*, vol. 20, pp. 1591-1609, 2008.
[106] S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Haldar, *et al.*, "Improving simulation for network research," 1999.
[107] E. Weingartner, H. Vom Lehn, and K. Wehrle, "A performance comparison of recent network simulators," in *2009 IEEE International Conference on Communications*, 2009, pp. 1-5.
[108] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, pp. 2899-2917, 2014.
[109] M. R. K. Grace, S. S. Priya, and S. Surya, "A Survey on Grid Simulators."
[110] H. Lamehamedi, Z. Shentu, B. Szymanski, and E. Deelman, "Simulation of dynamic data replication strategies in data grids," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, 2003, p. 10 pp.

[111] C. L. Dumitrescu and I. Foster, "GangSim: a simulator for grid scheduling studies," in *IEEE International Symposium on Cluster Computing and the Grid, 2005. CCGrid 2005.* , 2005, pp. 1151-1158.
[112] W. H. Bell, D. G. Cameron, A. P. Millar, L. Capozza, K. Stockinger, and F. Zini, "Optorsim: A grid simulator for studying dynamic data replication strategies," *International Journal of High Performance Computing Applications*, vol. 17, pp. 403-416, 2003.
[113] H. Casanova, "Simgrid: A toolkit for the simulation of application scheduling," in *Symposium on Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International* 2001, pp. 430-437.
[114] A. Legrand, L. Marchal, and H. Casanova, "Scheduling distributed applications: the simgrid simulation framework," in *Symposium on Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International* 2003, pp. 138-145.
[115] V.-D. Le, G. Babin, and P. Kropf, "BALLS Simulator: Evaluator of a structured Peer-to-Peer system with integrated load balancing," in *4th IEEE International Conference on Computer Sciences, Research, Innovation and Vision for the Future (RIVF)*, 2006, pp. 1-6.
[116] D. Klusáček, Š. Tóth, and G. Podolníková, "Complex Job Scheduling Simulations with Alea 4," in *Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques*, Prague, Czech Republic, pp. 124-129, 2016.
[117] Y. Huang, A. Brocco, M. Courant, B. Hirsbrunner, and P. Kuonen, "MaGate simulator: a simulation environment for a decentralized grid scheduler," in *International Workshop on Advanced Parallel Processing Technologies*, 2009, pp. 273-287.
[118] J. Kołodziej, S. U. Khan, L. Wang, M. Kisiel-Dorohinicki, S. A. Madani, E. Niewiadomska-Szynkiewicz, *et al.*, "Security, energy, and performance-aware resource allocation mechanisms for computational grids," *Future Generation Computer Systems*, vol. 31, pp. 77-92, 2014.
[119] F. Xhafa, L. Barolli, and D. Martos, "A web interface for the HyperSim-G Grid simulation package," *International Journal of Web and Grid Services*, vol. 5, pp. 17-29, 2009.
[120] D. Tsafrir and D. G. Feitelson, "The dynamics of backfilling: solving the mystery of why increased inaccuracy may help," in *2006 IEEE International Symposium on Workload Characterization*, 2006, pp. 131-141.
[121] R. Baraglia, G. Capannini, M. Pasquali, D. Puppin, L. Ricci, and A. D. Techiouba, "Backfilling strategies for scheduling streams of jobs on computational farms," in *Making Grids Work*, ed: Springer, 2008, pp. 103-115.
[122] D. Klusáček and H. Rudová, "Alea 2: job scheduling simulator," in *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, 2010, p. 61.
[123] M. Hassan and R. Jain, *High performance TCP/IP networking* vol. 29: Prentice Hall, 2003.
[124] D. Feitelson. (2005). *Parallel Workloads Archive*. Available: http://www.cs.huji.ac.il/labs/parallel/workload/
[125] *Workloads provided by the CERIT Scientific Cloud*. Available: https://metavo.metacentrum.cz/
[126] D. G. Feitelson, "Metrics for parallel job scheduling and their convergence," in *Workshop on Job Scheduling Strategies for Parallel Processing*, 2001, pp. 188-205.

[127] A. V. Gritsenko, N. G. Demurchev, V. V. Kopytov, and A. O. Shulgin, "Decomposition Analysis and Machine Learning in a Workflow-Forecast Approach to the Task Scheduling Problem for High-Loaded Distributed Systems," *Modern Applied Science,* vol. 9, p. 38, 2015.
[128] Z. Rizal, M. Azmi, T. Herawan, A. B. Kamalrulnizam, A. Abdul Hanan, and S. Mohd Shahir, "Combinatorial Rules Approach to Improve Priority Rules Scheduler in Grid Computing Environment," 2012.
[129] F. Xhafa and A. Abraham, *Meta-heuristics for grid scheduling problems*: Springer, 2008.
[130] M. G. A. Verhoeven and E. H. Aarts, "Parallel local search," *Journal of Heuristics,* vol. 1, pp. 43-65, 1995.
[131] K.-U. Stucky, W. Jakob, A. Quinte, and W. Süß, "Solving scheduling problems in Grid resource management using an evolutionary algorithm," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, 2006, pp. 1252-1262.
[132] D. Zotkin and P. J. Keleher, "Job-length estimation and performance in backfilling schedulers," in *High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on*, 1999, pp. 236-243.
[133] F. Xhafa, J. Kołodziej, L. Barolli, and A. Fundo, "A GA+ TS hybrid algorithm for independent batch scheduling in computational grids," in *Conference on Network-Based Information Systems (NBiS), 14th International*, 2011, pp. 229-235.
[134] F. Glover, G. A. Kochenberger, and B. Alidaee, "Adaptive memory tabu search for binary quadratic programs," *Management Science,* vol. 44, pp. 336-345, 1998.
[135] D. G. Feitelson, "Experimental analysis of the root causes of performance evaluation results: a backfilling case study," *IEEE Transactions on Parallel and Distributed Systems,* vol. 16, pp. 175-182, 2005.
[136] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "Selective reservation strategies for backfill job scheduling," in *Workshop on Job Scheduling Strategies for Parallel Processing*, 2002, pp. 55-71.
[137] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "Characterization of backfilling strategies for parallel job scheduling," in *Conference on Parallel Processing Workshops, 2002. Proceedings. International* 2002, pp. 514-519.
[138] L. Zhu and J. X. Tao, "Comparison of Stress Intensity Factors of Part Surface Flaws at Stress Concentration in Plate and Pipe Between FE Analysis and Weight Function Approach," in *ASME 2016 Pressure Vessels and Piping Conference*, 2016, pp. 1-9.
[139] X. Tang and S. T. Chanson, "Optimizing static job scheduling in a network of heterogeneous computers," in *International Conference on Parallel Processing, 2000. Proceedings in IEEE, 2000.*, 2000, pp. 373-382.
[140] L. Lamport, "How to make a multiprocessor computer that correctly executes multiprocess programs," *IEEE transactions on computers,* vol. 100, pp. 690-691, 1979.
[141] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, *et al.*, "Mapping abstract complex workflows onto grid environments," *Journal of Grid Computing,* vol. 1, pp. 25-39, 2003.
[142] R. Jain, "The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling," ed: John Wiley & Sons, 1990.

[143] Š. Tóth and R. L. Matyska, "Multifaceted Fairness in Job Scheduling Novel Fairness Model for HPC and Grids," Ph.D Dissertation, Faculty of Informatics, Universitas Masarykiana, Czech, 2014.
[144] J. Wolberg, *Data analysis using the method of least squares: extracting the most information from experiments*: Springer Science & Business Media, 2006.

Appendix A
Fairness in Swift Gap

The fairness applied in the Swift Gap (SG) mechanism (as a default feature) guarantees that different users in the system enjoy the same performance. Therefore, based on this concept of applied fairness, it can be concluded that this "factor" is related to the user rather than to the time spent on the job.

To improve fairness, the aim is to minimize the Normalized User Waiting Time values (NUWT). The closer the values of NUWTs to each other, the better the fairness among the users will be. To compute NUWT, two variables have to be calculated first: Total User Waiting Time (TUWT) and Total User Squashed Area (TUSA) [143].

Equation (1) calculates the TUWT for all users as follows:

$$TUWT = \sum_{j=1}^{J} (S_J - Sub_J) \quad (1)$$

Where S_J is the start time for the job and Sub_J is the submission time to the system for that job. Equation (2) calculates the TUSA for all users as follows:

$$TUSA = \sum_{j=1}^{J} (P_J * Usage_J) \quad (2)$$

Where $(P_J * Usage_J)$ is the job runtime and the number of requested resources.

Therefore, the Normalized User Waiting Time (NUWT) is as stated in Equation (3).

$$NUWT = \frac{TUWT}{TUSA} \tag{3}$$

To compute the User Waiting Time (UWT) for each user, (TUWT) is normalized for each user as explained in Equation (4).

$$TUSA = \frac{I}{u} \sum_{u=1}^{U} TUWT \tag{4}$$

Therefore, the Fairness (F) is as shown in Equation (5).

$$F = \sum_{u=1}^{U} (UWT - NUWT)^2 \tag{5}$$

The square guarantees all the obtained values are positive; this equation is based on least square method [144].

In the SG mechanism (explained in Chapter Four), F is considered as a criterion for acceptance. Thus, a job is moved based on F. F will be computed in both places and based on normal comparison, and the move will be accepted or rejected. After introducing the CTS as an acceptance criterion (in Chapter Five), F is kept (not debugged) in the optimization stage. The mechanism will compute the acceptance criterion first: if the move is rejected the job will stay; if the move is accepted F will be computed next. Based on the value of F, the move will be executed or cancelled.

Appendix B

Numerical Results

Chapter Four: Swift Gap Mechanism (Units are seconds, Slowdown is a fraction)

Mechanisms	*Slowdown*	*Bounded Slowdown*	*Tardiness*	*Waiting Time*	*Response Time*
SG (Zewura)	244.34	96.45	4501.77	8601.62	55126.1
EASY (Zewura)	657.71	215.50	7066.78	10965.01	57489.48
CONS (Zewura)	457.66	176.27	8777.39	14218.76	60743.23
SG (Wagap)	99.68	37.81	698.44	827.93	60605.09
EASY (Wagap)	181.64	55.7	936.39	1056.51	60830.91
CONS (Wagap)	155.54	50.41	740.69	879.62	60656.77
SG (Meta)	163.08	66.98	4085.35	5433.40	23390.86
EASY (Meta)	200.51	80.84	4935.74	6468.78	24426.24
CONS (Meta)	198.12	79.317	5113.73	6715.30	24672.76

Chapter Five: Completion Time Scheme

Mechanisms	*Slowdown*	*Bounded Slowdown*	*Tardiness*	*Waiting Time*	*Response Time*
SG (Zewura)	244.34	96.45	4501.77	8601.62	55126.1
CTS (Zewura)	222.78	84.67	4365.34	8338.20	53150.67
SG (Wagap)	99.68	37.81	698.44	827.93	60605.09
CTS (Wagap)	67.61	25.5	381.75	523.12	59378.28
SG (Meta)	163.08	66.98	4085.35	5433.40	23390.86
CTS (Meta)	159.57	66.39	3916.03	5270.97	19978.44

Chapter Six: Swift Gap with Completion Time Scheme

Mechanisms	*Slowdown*	*Bounded Slowdown*	*Tardiness*	*Waiting Time*	*Response Time*
SG-CTS (Zewura)	222.78	84.67	4365.34	8338.20	53150.67
CONS-GS (Zewura)	430.06	145.63	5140.89	9051.83	55576.30
Flex-CONS (Zewura)	267.18	112.48	6080.85	10133.95	56658.42
SJF (Zewura)	1048.69	520.03	30007.20	41950.66	88475.13
SG-CTS (Wagap)	67.61	25.5	381.75	523.12	59378.28
CONS-GS (Wagap)	97	37.61	699.57	830.16	60607.31
Flex-CONS (Wagap)	106.5	39.7	765.40	890.98	60668.14
SJF (Wagap)	173.28	64.9	922.12	1189.93	60967.08
SG-CTS (Meta)	159.57	66.39	3916.03	5270.97	19978.44
CONS-GS (Meta)	164.45	67.74	4070.63	5413.62	23371.08
Flex-CONS (Meta)	172.84	71.81	5637.41	6970.72	24928.18
SJF (Meta)	761.52	181.81	10668.36	13899.78	31857.23

Appendix C
The Data Fields in Standard Workload Format

1. **Job Number:** a counter field, starting from 1.

2. **Submit Time:** in seconds. The earliest time the log refers to is zero and is usually the submittal time of the first job. The lines in the log are sorted by ascending submittal times. It makes sense for jobs to also be numbered in this order.

3. **Wait Time:** in seconds. The difference between the job's submit time and the time at which it actually began to run. Naturally, this is only relevant to real logs, not to models.

4. **Run Time:** in seconds. The wall clock time the job was running (end time minus start time).

5. **Number of Allocated Processors:** an integer. In most cases, this is also the number of processors the job uses; if the job does not use all of them, we typically do not know about it.

6. **Average CPU Time Used:** both user and system, in seconds. This is the average over all processors of the CPU time used and may therefore be smaller than the wall clock runtime. If a log contains the total CPU time used by all the processors, it is divided by the number of allocated processors to derive the average

7. **Used Memory:** in kilobytes. This is again the average per processor.

8. **Requested Number of Processors.**

9. **Requested Time:** This can be either runtime (measured in wallclock seconds), or average CPU time per processor (also in seconds) -- the exact meaning is determined by a header comment. In many logs this field is used for the user runtime estimate (or upper bound) used in backfilling. If a log contains a request for total CPU time, it is divided by the number of requested processors.

10. **Requested Memory:** (again kilobytes per processor).

11. **Job Status:** Status 1 if the job was completed, 0 if it failed and 5 if cancelled. If information about chekcpointing or swapping is included, other values are also possible.

12. **User ID:** a natural number, between one and the number of different users.

13. **Group ID:** a natural number, between one and the number of different groups. Some systems control resource usage by groups rather than by individual users.

14. **Executable (Application) Number:** a natural number, between one and the number of different applications appearing in the workload. in some logs, this might represent a script file used to run jobs rather than the executable directly; this should be noted in a header comment.

15. **Queue Number:** a natural number, between one and the number of different queues in the system. The nature of the system's queues should be explained in a header comment. This field is where batch and interactive jobs should be differentiated: we suggest the convention of denoting interactive jobs by 0.

16. **Partition Number:** a natural number, between one and the number of different partitions in the systems. The nature of the system's partitions should be explained in a header comment. For example, it is possible to use partition numbers to identify which machine in a cluster was used.

17. **Preceding Job Number:** this is the number of a previous job in the workload. If -1 value is existed, no precedence whatsoever.

18. **Think Time from Preceding Job:** this is the number of seconds that should elapse between the termination of the preceding job and the submittal of this one.

Example

475 1325189640 4063 46244 16 1 1024 16 345600 68719476736 1 836 1 1 0 1 -1 1

1. **Job Number:** 475
2. **Submit Time:** 1325189640,
3. **Wait Time:** 4063
4. **Run Time:** 46244
5. **Number of Allocated Processors:** 16
6. **Average CPU Time Used:** 1
7. **Used Memory**: 1024
8. **Requested Number of Processors:** 16
9. **Requested Time:** 345600
10. **Requested Memory:** 68719476736
11. **Job Status:** Completed
12. **User ID:** 836
13. **Group ID:** 1
14. **Executable (Application) Number:** 1
15. **Queue Number**: Interactive (0)
16. **Partition Number**: 1
17. **Preceding Job Number**: -1
18. **Think Time from Preceding Job**: 1