AN EFFECTIVE JOB SCHEDULING ALGORITHM IN HIGH PERFORMANCE COMPUTING

LETT YI KYAW

UNIVERSITY OF COMPUTER STUDIES, YANGON

JUNE, 2024

An Effective Job Scheduling Algorithm in High Performance Computing

Lett Yi Kyaw

University of Computer Studies, Yangon

A thesis submitted to the University of Computer Studies, Yangon in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

June, 2024

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

…..…… ……..……

Date Lett Yi Kyaw

ACKNOWLEDGEMENTS

First and first, I would like to express my gratitude to His Excellency, the Minister of the Ministry of Science and Technology, for providing full facility support for the University of Computer Studies, Yangon Ph.D. course.

Secondly, I would like to thank the University of Computer Studies, Yangon's Rector, Dr. Mie Mie Khin, in particular for helping me with this thesis and for her general advice during my studies.

Additionally, I would want to express my gratitude and special appreciation to Dr. Sabai Phyu for her insightful remarks, counsel, and guidance, all of which have been very helpful to me.

I also want to express my gratitude and special appreciation to Dr. Tin Thein Thwe for her kindness, insightful guidance, and helpful comments, all of which have been very helpful to me.

I sincerely thank Dr. Si Si Mar Win, a professor at the University of Computer Studies in Yangon, who is the course coordinator. She has been very patient, kind, and supportive of my research efforts both morally and emotionally.

I would like to acknowledge my supervisor, Dr. Kyar Nyo Aye, Associate Professor, University of Computer Studies, Taunggyi, from the bottom of my heart for all of her help, support, and patience. She also gave me great ideas for my research projects.

I would like to express my respectful gratitude to Daw Aye Aye Khine, Professor, Head of English Department for her valuable supports from the language point of view and pointed out the correct usage in my dissertation.

I also thank my friends from Ph.D. 11th batch for providing support and friendship that I needed.

Last but not least, I am very much indebted to my parents and my younger brother for always believing in me, for their endless love and support. They are always supporting and encouraging me during the years of my Ph.D. study.

ABSTRACT

The need for efficient resource management in distributed computing systems is greater than ever due to the significant growth of High Performance Computing (HPC) and its predicted future expansion. The increasing volume of computing workloads requires the application of effective resource management and workload scheduling in order to maximize resource utilization while maintaining a reasonable level of computational performance. It is difficult to effectively schedule workloads on resources while maintaining performance requirements. Furthermore, resource management becomes much more difficult in real-world situations due to non-clairvoyance regarding task dimensions. Using the latest machine learning and data science tools, the suggested study methodology explores the scheduling problem that is compatible for HPC and explores the difficulties in implementing the scheduling in real-world scenarios. The majority of recent scientific and engineering advancements are attributed to high performance computing. High performance systems, which are massive parallel networked supercomputers, are necessary for research conducted at national laboratories. In a world where data is becoming more and more important, High Performance Computing (HPC) systems are becoming essential tools for solving complex, compute-intensive, and data-intensive problems in a variety of engineering, business, and scientific domains. This has led to new discoveries in science and technology as well as the development of more dependable and effective goods and services. In High Performance Computing (HPC) platforms, job scheduling is a challenging problem with unknowns regarding the arrival process and execution time of jobs. This proposed system is try to improve Min-min and priority algorithm based on combination of existing two algorithms. Various methods are searching for scheduling using High Performance Computing (HPC) in research trend. The proposed system will show the comparison among Min-min, priority and hybrid approach for job scheduling system. The performance of different HPC systems are evaluated by Quality of Service (QoS) metrics (turnaround time, throughput and response time). To test the effectiveness and correctness of simulator, experiments will run to gather statistics. The workload logs used in the experiments are the HPC2N log, KIT-FH2-2016-1log, and the SDSC-SP2-1998-4.2-cln log given in the Parallel Workloads Archive.

Table of Contents

Acknowledgements i

Abstract ii

Table of Contents iii

List of Figures vi

List of Tables viii

List of Equations ix

1. INTRODUCTION 1

1.1 Architecture of HPC……. 2

1.2 Scheduling Algorithms of HPC……. 3

1.3 Motivation of the Thesis……..…….. 4

1.4 Problem Statement……. …… 4

1.5 Objectives of the Thesis…….…….. 5

1.6 Contributions and Research Direction ……. 5

2. LITERATURE REVIEWS 6

2.1 Utility Computing……. …… 7

2.2 Cluster Computing …… 7

2.3 Grid Computing ……. 8

2.4 Cloud Computing……. ……. 9

2.5 High Performance Computing (HPC) …… ….…….. 11

2.6 Fog Computing ……..……. 12

2.7 Difference between Cloud Computing and HPC …….. 12

2.8 Related Works…….. ……. 14

2.9 Summary…… 20

3. BACKGROUND THEORY 21

3.1 Evolution of High Performance Computing……. 21

3.2 HPC System ……...…… 23

3.3 Job Scheduling ……...…… 27

3.3.1 Job Types ……...…… 27

3.3.2 Scheduling Metrics Scheduling Policies...…… 29

3.3.2.1 System Throughput …….. 29

3.3.2.2 Makespan …… …….. 30

3.3.2.3 Waiting Time …… …….. 30

3.3.2.4 Turnaround Time …….. 30
3.3.2.5 Reaction Time …… …….. 31
3.3.2.6 Response Time …… …….. 31
3.3.2.7 Fairness …… …….. 31
3.3.2.8 Goodput …… …….. 32
3.3.3 Scheduling Policies ……..…….. 32
3.3.3.1 Ordering Policy …….. 32
3.3.3.2 Resource Selection Policy…..…….. 33
3.3.3.3 Reservation Policy …….. 33
3.4 Scheduling Algorithms ……...…… …… 34
3.4.1 First Come First Served (FCFS) ……...……. 34
3.4.2 Shortest Job First…..……...……..... 34
3.4.3 Round Robin (RR) ..……...…… 35
3.4.4 Priority Scheduling ……...…… 35
3.4.5 Multilevel Queue Scheduling……...……. 35
3.4.6 Multilevel Feedback Queue Scheduling...…… 36
3.4.7 Fair Share Scheduling ……...…… 36
3.4.8 Min-Min Scheduling …….……...…… 36
3.4.9 EASY Backfilling Scheduling ……...…… 36
3.5 Simulation …….…….……. ……. 36
3.6 Summary …… 38
4. THE PROPOSED SYSTEM ARCHITECTURE 39
4.1 The Proposed System Architecture …… ……... 39
4.2 Proposed System ……..... 41
4.3 Summary ……..... 44
5. DESIGN AND IMPLEMENTATION OF THE PROPOSED SYSTEM 45
5.1 Input The Proposed System 45
5.2 Graphical User Interface of the Proposed System……. 50
5.3 Summary ……. 51
6. EXPERIMENTAL RESULTS 53
6.1 Experimental Results……..... 53
6.1.1 FCFS ……..... 55
6.1.2 Proposed System 57

6.1.3 Easy Backfilling..... 58
6.2 Comparison Results……..... 60
6.3 Summary……..... 60
7. CONCLUSION AND FUTURE WORK 61
7.1 Summary..... 61
7.2 Discussion..... 62
7.3 Advantages and Limitations of the Proposed System…….. 62
7.4 Future Work..... 62
Author' Publications……..... 65
Bibliography……. 66

List of Figures

3.1	Timeline of HPC Evolution ……..…..……...	22
4.1	Pseudo-Code for Min-Min Algorithm……	40
4.2	Proposed System Architecture ……. ….……...	41
4.3	Flow Chart of The Proposed System ……	42
4.4	Proposed Job Scheduling Algorithm……..	43
5.1	Workload Log Files ……..……	46
5.2	HPC2N.swf workload file ……	48
5.3	KIT-FH2-2016-1.swf workload file……..,….…….	48
5.4	SDSC-SP2-1998-4.2-cln.swf workload file ……..…….	49
5.5	Simulation Result ……	50
5.6	The Simulator ……..…….	51
6.1	Throughput [FCFS, using hpc2n.swf data set] …….	55
6.2	Turnaround Time [FCFS, using hpc2n.swf data set]…….....	55
6.3	Response Time [FCFS, using hpc2n.swf data set] …….	56
6.4	Throughput, Turnaround Time ,Response Time [proposed algorithm, using hpc2n.swf data set]……	56
6.5	Throughput [proposed algorithm, using hpc2n.swf data set] …….	57
6.6	Turnaround Time [proposed algorithm, using hpc2n.swf data set] …..	57
6.7	Response Time [proposed algorithm, using hpc2n.swf data set]…….	58
6.8	Throughput [EASY Backfilling, using hpc2n.swf data set] …….	58
6.9	Turnaround Time [EASY Backfilling, using hpc2n.swf data set]……..	59

6.10 Response Time [EASY Backfilling, using hpc2n.swf data set]…….. 59

List of Tables

3.1 HPC System …….. 26

5.1 Data Fields in Data Sets …… 47

6.1 Comparison Results of FCFS, EASY Backfilling, Proposed Algorithm 60

List of Equations

Equation 6.1 …….. …….. 54

Equation 6.2 …….. 54

CHAPTER 1
INTRODUCTION

The majority of recent scientific and engineering advancements are attributed to high performance computing. High performance systems, which are massive parallel networked supercomputers, are necessary for research conducted at national laboratories. High Performance Computing (HPC) systems are used in research in scientific domains including bio-informatics, computational chemistry, and computational physics to execute intricate, large-scale computational operations. A job is an application used by a user that might need one (sequential) or multiple (parallel) CPUs. There are predetermined arrival and work processing times. Every task has a proprietor. One or more computer clusters, each consisting of multiple machines, make up the system. The quantity and speed of CPUs shared by all machines in a cluster are fixed parameters. Every computer in a cluster that permits the cluster to execute multiple jobs in parallel when the sum number of CPUs desired is equivalent to or less than the cluster's CPU count.

Since the year 2000, the globe has produced an enormous amount of data, largely due to the growth of the Internet. In 2010, the monthly global internet traffic was approximately 20 EB (Extended Benefits (EB)). However, in 2017, that number had increased to almost 120 EB (6 times in 7 years). The conventional method of storing data on a database, which is housed on a single large system, is no longer appropriate. The process must be spread throughout the entire infrastructure because to the volume, variety, and speed of data output. To address this issue, massive Web corporations gave rise to the field of big data.

Large-scale, dispersed data collection, storing, indexing, and analysis are required via a network of related instruments. However, it also sees the exponential increase in processing power of the largest computers in the world, known as supercomputers. With regard to upcoming HPC devices, the exascale, a scale at which a single supercomputer can compute at the exaFLOPS (1018 floating operations per second) level.

Computing has advanced faster and more consistently than any other technology; supercomputers of today surpass those of the 1950s by a trillion (1,000,000,000,000) times speed. Supercomputer performance is expressed in terms

of flops, which is an acronym for floating point operations per second, which represents the computer's ability to process mathematical computations quickly. Today's most potent supercomputers are capable of one quadrillion (1,000,000,000,000,000) flops, or petaflop performance. With almost half of the world's top 500 supercomputers located here and 90% of them being constructed by American hardware companies, the United States leads the world in high-performance computing. A significant portion of the advancements in computers can be credited to the Department's work.

One of the interconnected aspects of computers is task scheduling. Task scheduling faces additional difficulties as a result of the Grid and ubiquitous computing, which are based on factors like security, quality of service (QoS), and the absence of central control in distributed administrative domains.

The majority of jobs that are submitted to HPC systems are parallel programs with several running stages, including data input/output, calculation, and communication. As a result, executing these concurrent apps may require a variety of resources, including power, storage, network, and I/O bandwidth. Also, the majority of these system resources are shared by several jobs that are executing at once. Algorithms for job scheduling are being studied in numerous research simulation processes. Before using a scheduling algorithm in a production system, it must undergo extensive testing and assessment in its development or application. A crucial component of distributed and parallel computing is job scheduling. In this field, a great deal of study has been done, producing important theoretical and practical outcomes. On the other hand, new scheduling methods are required to handle issues arising from the Grid architecture with the rise of the computational Grid. The scheduling difficulty in a supercomputer center is compounded by the need to maximize system utilization while scheduling a collection of applications from various users to the parallel machine. A scheduling algorithm should aim to enhance system usage, boost throughput, and satisfy user and system limitations while remaining inexpensive.

1.1 Architecture of HPC

Each of the many identical nodes that make up the HPC cluster contains many cores, some memory, and some disk space. A job that is submitted to the cluster asks

for some memory, disk space, one or more nodes, and one or more cores per node. This type of scheduling, in which every machine or processor is the same, is known as "Scheduling on Multiple Machines" or "Multiprocessor Scheduling" in scheduling theory.

In HPC, job scheduling and resource management are crucial. Efficient job/resource scheduling algorithms have been the subject of extensive inquiry in recent years. When working on computer clusters, process scheduling has been considered one of the most crucial topics to focus on because a cluster's productivity increases when a process scheduling algorithm performs better.

In general, task scheduling difficulties require choosing a resource for each job and a job processing order/time based on a variety of restrictions. Different jobs may require different resources (CPU, memory, storage space, network transportation, etc.) and different levels of priority. Therefore, in order to handle job scheduling challenges, it is necessary to get a better scheduling algorithm. This algorithm may be evolved from the current approaches by adding more metrics, which can lead to good performance and outputs that can be used in real-world scenarios down the road.

The term "node" refers to five distinct types of nodes: computational, storage, control, management, and user.

User Node: The sole point of entry into the cluster system for outsiders. In order to compile and execute the tasks, users typically need to log in from the node.

The control node is primarily in charge of providing the computing node with essential network services, like DNS, NFS, and DHCP (Dynamic Host Control Protocol), as well as assigning duties to the computing node.

1.2 Scheduling Algorithms of HPC

Because scheduling in HPC environments is always done online, there are additional levels of complexity. Decisions concerning scheduling are made in an online environment without full knowledge of the tasks that need to be scheduled. There are two main areas where this ignorance manifests itself: the arrival time as well as the job execution (processing) time. (i) Arrival times: Because work can arrive at any time, scheduling decisions must be decided "on the fly," without considering future assignments. (ii) Processing times: The exact amount of time that a task will take to complete is not known in advance and can only be found out after it has been

completed. Precise runtimes are frequently replaced by inaccurate user-provided estimates.

1.3 Motivation of the Thesis

In HPC, job scheduling and resource management are crucial. Efficient job scheduling algorithms have been the focus of a significant amount of research in recent years. Choosing a work processing order for each resource, given a variety of limitations, and choosing a resource for each job include the general job scheduling challenges. Different jobs may require different resources (CPU, memory, storage space, network transportation, etc.) and different levels of priority.

Each of the many identical nodes that make up the HPC cluster contains many cores, some memory, and some disk space. A job that is submitted to the cluster asks for some memory, disk space, one or more nodes, and one or more cores per node. This type of scheduling, in which every machine or processor is the same, is known as "Scheduling on Multiple Machines" or "Multiprocessor Scheduling" in scheduling theory. The main goal is to minimize (or maximize) a criterion while scheduling a job from a list of available jobs on one or more machines. A scheduling method is essential to enhancing the computer's server and resource performance. In recent times, complicated issues such as scientific applications have become more complex due to the integration of multiple approaches and techniques into a single solution. Therefore, more advanced scheduling algorithms are required to address task scheduling issues. These can be created from the two existing scheduling algorithms and compared using metrics that will produce high-quality outputs that may eventually be used in real-world settings.

1.4 Problem Statement

HPC requires a large number of computers to efficiently handle several jobs at once. Various scheduling algorithms are used, and performance metrics are used to determine which approach works best based on resource requirements. Various scheduling strategies have been compared based on performance factors like as throughput, CPU utilization, response time, and scalability for HPC systems. Each algorithm handles distinct challenges and yields different results. The suggested

solution aims to enhance a system's throughput, turnaround time, and response time, or QoS. Starvation and waiting times are expected to decrease under the suggested system.

1.5 Objectives of The Thesis

This research's primary goal is to minimize load balancing. The following are the research's other goals:

(i) To allocate suitable resources to workflow jobs so that the application gets execute to completion and at the same time the user requirements are satisfied.

(ii) To present a hybrid scheduling algorithm that combines the advantages of the priority and min-min scheduling strategies.

(iii)To show the results of Throughput, Turnaround and Response Time in different data sets and different scheduling algorithms.

1.6 Contributions and Research Direction

Nowadays, the HPC is important research area. It is widely accepted in most industries, such as financial services and healthcare. High Performance Computing (HPC) requires a large number of computers to efficiently handle several jobs at once. Various scheduling algorithms are used, and performance metrics are used to determine which approach works best based on resource requirements. Various scheduling strategies have been compared based on performance factors like as throughput, CPU utilization, response time, and scalability for HPC systems. Each algorithm handles distinct challenges and yields different results. The suggested solution aims to enhance a system's throughput, turnaround time, and response time, or QoS. Starvation and waiting times are expected to decrease under the suggested system.

The introduction to HPC, key performance measures, issue descriptions, aims, motivations, and contributions of the research study are among the seven chapters that make up this dissertation. The second chapter examines the various HPC systems in the literature that are relevant to the dissertation. Chapter 3 provides a description of HPC's theoretical foundation. Chapter 4 discusses the suggested system design and

HPC algorithm recommendations. Chapter 5 presents the suggested algorithm's design and implementation. The assessment of the experimental results using processing time and HPC quality measurement metrics is covered in Chapter 6. Ultimately, Chapter 7 summarizes the research work's conclusion and outlines potential directions for future investigation.

CHAPTER 2

LITERATURE REVIEWS

The application of high performance computing, or HPC, is now essential for industry and science to advance their fields and find new information. Supercomputers were exclusively available to the greatest multinational organizations and institutions, such as the military, a few decades ago. However, during the previous few years, the cost of computing power has decreased dramatically due to ongoing technological advancements and improvements in production methods, creating new, more diverse markets and enabling supercomputing to be accessed by a wider range of academic, commercial, and institutional domains. These days, HPC is used in practically every field in one way or another.

Supercomputing is the use of powerful, massively parallel computers to tackle complex computational tasks that would take years or even decades for a conventional computer to finish, as the name suggests. Supercomputing is pervasive in practically every facet of our daily existence. Our lives can be altered by supercomputers in a number of ways, including increased industry competition, positive health effects, improved forecasting, the ability to make more specialized advancements, and more trustworthy decision-making. Supercomputers are frequently used to evaluate mathematical models that describe complex physical phenomena or designs, such as climate and weather, cosmic evolution, nuclear weapons and reactors, innovative chemical compounds (especially for practical applications), and cryptology.

Today, the automotive, petroleum, and aerospace industries commonly use high-performance computing. A growing number of companies began using supercomputers for business-related applications such as market research in the 1990s as the cost of supercomputing decreased. They were also used for playing video games online. Notably, in 2007 a business holding online rights to environment of Warcraft, which at one point hosted over a million people concurrently in the same virtual environment, possessed the fifth to tenth fastest supercomputers in China. Supercomputers are different from regular computers in that they have special qualities. Typically, they have several CPUs, or central processing units, with circuits inside of them that decode instructions from programs and sequentially carry out arithmetic and logic operations. Because circuit technology has physical limitations,

large computing rates require the employment of several CPUs. Since electrical signals are limited to the speed of light, this establishes a fundamental speed constraint for circuit switching and signal transmission. This limit is almost reached due to the miniaturization of circuit components, significant reductions in the length of wires connecting circuit boards, and advancements in cooling techniques (e.g., Cold fluid is used to reach the low temperatures at which processor and memory circuits in various supercomputer systems operate most efficiently). This chapter's first portion examines the literature on cluster computing, HPC, cloud computing, grid computing, and utility computing. Next, the differences between HPC and the cloud are discussed in relation to schema conversions.

2.1 Utility Computing

Utility computing is a service provisioning approach that provides clients with on-demand, as-needed access to computer resources. Charges are based precisely on the amount of services used rather than a fixed or flat rate. It is a kind of cloud computing where users can change how they use resources based on what they require. Services including data storage, processing power, application services, virtual servers, and even hardware rentals like CPUs, displays, and input devices are available to customers, users, and organizations.

The utility computing concept, which makes IT resources as easily accessible as public utilities like electricity, gas, water, and phone services, is modeled after traditional utilities. For example, a consumer of electricity pays for the number of units used, neither more nor less. The utility computing architecture, which gives each job an economic feature called utility and defines this utility through a Time Utility Function (TUF), is frequently used in cloud computing to handle job scheduling. The authors of [23] [24] created a scheduling algorithm that is fault-aware and efficient, with an emphasis on long-term optimization, and it is based on the Reinforcement Learning (RL) framework. For the practical deployment of commercial cloud services, there is currently no well-defined job scheduling algorithm that considers future system states, especially under overload conditions.

2.2 Cluster Computing

A cluster is made up of separate IT resources joined together to perform as a single unit. By combining redundancy and failover capabilities, this configuration

lowers the rate of system failure while increasing system availability and reliability. A cluster is essentially an assembly of several computers, or nodes. The master node, the single node that is immediately accessible from the network and acts as the cluster's public interface, is accessed by users who log into a cluster. Through an internal private network, the other nodes—storage and computing nodes included—communicate with the master node and with each other.

All cluster users share access to the master node, which is utilized for file transfers as well as job preparation, submission, and management. It is crucial to avoid using the master node for computationally demanding operations. Jobs really run on one or more of the compute nodes, even though they are submitted from the master node. On the compute nodes, job scheduling and allocation are managed by the resource manager and job scheduler. IBM Platform Computing is the resource management used by Vikram-100.

Jobs can be batch or interactive and are submitted using a queue submission command. For testing and debugging purposes, an interactive job offers a login session on a compute node that permits direct interaction with the node by issuing instructions throughout the session. On the other hand, a batch job is a written job that is intended for production runs lasting several hours or days. It executes through to completion without the need for user involvement. The majority of jobs on the cluster are batch tasks.

To ensure similar performance levels, the hardware components of a cluster typically need to have reasonably identical hardware and operating systems, facilitating the replacement of failed components. The devices forming a cluster are kept in sync through dedicated, high-speed communication links.

2.3 Grid Computing

A computing grid, or "computational grid," organizes computing resources into one or more logical pools that work together to form a high-performance distributed grid, often called a "super virtual computer." Unlike clusters, grid systems are more loosely coupled and distributed, allowing them to utilize heterogeneous and geographically dispersed resources, which is typically not possible with cluster-based systems. Research into grid computing has been ongoing since the early 1990s, influencing various aspects of cloud computing platforms, especially features like

networked access, resource pooling, scalability, and resilience. Both grid and cloud computing can establish these features, but they do so in distinct ways.

In the past ten years, hundreds of applications—the majority of which are in early prototype form—have been presented to grid infrastructures from academics, commerce, and industry. Each application is unique, solving specific problems by modeling phenomena (such as in physics, chemistry, or biology) with mathematical formulas, numerical methods, programming languages, and computer architectures. These applications are embedded in workflows and accessed remotely through secure, application-specific portals, highlighting the complexity of problem-solving on grid infrastructures.

Developers encounter many levels of complexity when porting software to a computer environment, especially to a compute or data grid with dispersed networked nodes. These nodes range from desktops to supercomputers, typically containing several loosely or tightly coupled processors with many cores. Adjusting to various degrees of granularity—from the coarse granularity of application workflows in multi-physics applications to fine-grain structures using multi-core architectures—is necessary for efficient application performance on such systems. Developers also need to take into account certain grid requirements, such as data management, information services, security, and resource management. The grid is sometimes seen as a replacement for supercomputers due to its massive CPU cycle availability. However, this view is flawed. The purpose of a supercomputer is to solve complex problems quickly by focusing substantial resources on a single issue. While the grid is effective for embarrassingly parallel problems that can be divided into small, independent tasks (as seen with projects like Berkeley Open Infrastructure for Network Computing and the World Community Grid), it struggles with large-scale problems that cannot be easily split. Such problems require the focused power of a supercomputer to be solved in a practical timeframe, which the grid cannot provide if it aims to replace supercomputers with distributed systems.

2.4 Cloud Computing

Delivering a variety of computer services via the internet (the cloud) includes servers, storage, databases, networking, software, and more. This approach promotes quick innovation, adaptable resource availability, and scalability benefits. By

removing the need for businesses to purchase and maintain actual hardware, this paradigm improves accessibility, scalability, and reduces costs and complexity. Resource provisioning and scheduling are two aspects of resource allocation, which is the process of allocating virtual machines (storage, processing, and networking) to cloud users' applications.

Cloud computing is generally divided into three service models:

Infrastructure as a Service (IaaS): Provides virtualized computing resources over the internet. Users can rent virtual machines, storage, and networking infrastructure on a pay-as-you-go basis.

Platform as a Service (PaaS): Offers a platform that allows users to design, run, and administer applications without dealing with the difficulties of creating and maintaining the underlying infrastructure. Typically, databases, middleware, development tools, and other services are included.

Software as a Service (SaaS): Delivers software applications over the internet on a subscription basis, enabling users to access the software through a web browser without needing to install or maintain it locally.

Launched in beta in August 2006, Amazon Elastic Compute Cloud (EC2) is a fundamental component of Amazon Web Services (AWS), the company's cloud computing platform. Applications can be executed on Virtual Machines (VMs) that users can rent. Scalable application deployment is made possible by EC2 because it offers a web service that lets customers boot an Amazon Machine Image (AMI) and turn it into a virtual machine, or "instance," that runs any software they want. Users can pay by the hour for active servers and create, activate, and terminate server instances as needed. Users can choose where their instances are located, and EC2 optimizes latency while offering high levels of redundancy.

In order to assess whether cloud services are suitable for HPC applications, [20] ran an HPC benchmark on Amazon EC2. He compared clusters of cutting-edge CPUs from Amazon EC2 with an HPC cluster at the National Center for Supercomputing Applications (NCSA) using a number of macro and micro benchmarks. NAS Parallel Benchmarks [35] were employed by him to evaluate these clusters' performance for standard scientific computations. Results from [20] also demonstrated MPI performance in these clusters using the moppets micro benchmark, which is noteworthy considering the importance of the Message-Passing Interface (MPI) library in scientific computing. He made use of EC2's high-CPU extra-large

instances for his research. Eight programs—five parallel kernels and three simulated applications—are included in the NAS Parallel Benchmarks [35], a collection of programs frequently used to assess the performance of HPC systems. The critical processing and data movement involved in computational fluid dynamics and other common scientific computations are collectively replicated by this benchmark suite.

According to research on the runtimes of each NPB program, programs operating on EC2 nodes performed worse than those running on the NCSA cluster compute nodes by between 7%–21% in [20]. Subsequent investigation showed that EC2 compute nodes' message-passing latencies and bandwidth were noticeably worse than those of the NCSA cluster. [20] came to the conclusion that if a high-performance network provisioning solution could be created to deal with this problem, the HPC scientific community may gain a lot.

2.5 High Performance Computing (HPC)

The term high-performance computing (HPC) describes the application of parallel processing methods and supercomputers to the resolution of challenging computational issues that call for substantial memory and processing power. In HPC systems, several networked processors collaborate to execute calculations at speeds significantly faster than those of conventional computer systems. As the name suggests, supercomputing is the application of strong, massively parallel computers to solve difficult computational problems that would take years or even decades for traditional computers to handle. Supercomputing is pervasive in practically every facet of our daily existence. ranging from medicinal applications to automobile designs to weather forecasting. Weather applications include aircraft flight simulations, natural catastrophe prediction, and forecasts—possibly the most well-known use of supercomputers. Supercomputers are used in practically every aspect of the biomedical industry. Molecular modeling, genetic sequencing, and drug creation through chemical reaction simulations are just a few of the uses for these machines. HPC is helping automotive designers by enabling high-fidelity crash simulations, new materials behavior simulation, and topological optimization, all while cutting costs and speeding up the production and test cycle.

There are big developments happening in HPC. Applications that are both compute- and data-intensive are included in the growing HPC applications. Most HPC

schedulers now in use are CPU-centric. System managers can better utilize all available resources by exploring possible tradeoffs between different resources thanks to the varied solutions that BBSched generates. In order to demonstrate how BBSched can be expanded to include other resources, the authors [91] provided a case study.

In order to achieve high processor utilization, high throughput, and a low reaction time, the processor scheduling method FCFS is implemented on HPC utilizing MPI [79].

2.6 Fog Computing

A type of distributed computing known as fog computing moves data storage and processing closer to the network edge, which is where a lot of Internet of Things devices are situated. Fog computing does this by decreasing the need for the cloud to handle these resource-intensive operations, which enhances performance and lowers latency. By moving processing and data storage closer to the edge, mist computing expands on cloud fog computing. Cloud computing servers, which are low-power servers that can be set up in huge quantities, are frequently used in this process.

2.7 Difference between Cloud Computing and High Performance Computing (HPC)

High performance computing and the cloud were once assumed to be synonymous. However, they aren't the same in reality. Cloud computing encompasses a wide range of applications, from complex computation to routine operations. It can be used for tasks that are specific to it as well as ones that you could perform on a typical home computer. As the name implies, high performance computing is essentially conventional computing on a much larger scale. Three billion calculations could be performed by a normal home computer every second. However, an HPC system can perform quadrillions of calculations every second. HPC is possible through parallel processing, in which a task is completed by a number of nodes—up to thousands—working together to complete it. Usually, these jobs entail data manipulation and analysis.

People frequently inquire, "Will HPC codes move to the cloud?" or "Are Grids dead now that cloud computing is widely accepted?" or even "Should my Grid be built in the cloud now?" Many feel perplexed and reluctant to move forward in spite

of all the encouraging advancements in the Grid and cloud computing area, as well as the increasing number of publications and conferences on the topic. This uncertainty is being caused by several factors, which are covered in the sections that follow.

As some experts had predicted, grids did not develop into the next essential IT infrastructure for mainstream HPC. Different middleware architectures (for department, enterprise, global, computation, data, sensors, devices, etc.) had to be designed and faced with varying usage models with varying benefits due to the diversity of computing environments. Global Grids are ideally suited for resource sharing and sophisticated R&D application collaboration, whereas HPC Grids offer superior resource usage and flexibility. Grid setup and operation was typically too complex for enterprise use. This feature—that is, the difficulty of building complicated HPC applications—was viewed by R&D specialists as an unavoidable disadvantage.

Grid computing was undoubtedly the next great thing for large science researchers and huge challenges after 40 years of working with HPC; yet, for enterprise CIOs, the Grid was just a stopover on their route to the cloud model. Private and public clouds today provide businesses all the components they need to succeed: they are simple to use, offer economies of scale, allow for business elasticity in ups and downs, and allow for pay-as-you-go, which eliminates some capital expenditure (CapEx) while still addressing the previously listed obstacles. Additionally, there is always the private cloud option when security is a concern. Private clouds may readily link to public clouds to form a hybrid cloud infrastructure in more complicated HPC environments where applications are operating under multiple regulations. This helps to balance security with elasticity and efficiency.

Every HPC simulation task is unique. Work varies according to IP, licenses, deadlines, budget, strategic significance, and priority. Furthermore, a particular computer architecture, operating system, memory, and other resources are frequently required due to the nature of the code. These significant variables affect the location and timing of an employment. Any new job type requires a set of requirements that determine which policies need to be developed and programmed into the scheduler in order for any of these jobs to operate in accordance with those policies. A dynamic resource broker that manages submission to Grid or cloud resources—whether local or global, private or public—ideally ensures this.

Low latency and large bandwidth are not necessary for many HPC applications. In science and engineering, parameter studies, or sweeps, are commonplace. They involve executing a single application for a range of parameters, leading to numerous independent tasks. Some examples include analyzing data from particle physics colliders, determining the solution parameter in numerical optimization, conducting ensemble runs to quantify uncertainties in climate models, identifying potential drug targets by screening compound structure databases, researching the sensitivity of economic models to parameters, simulating flow around airplane wings with varying angles of attachment, and assessing various materials and their resistance in crash tests, to mention a few.

Numerous sophisticated grand challenge research and engineering applications exist, according to the DEISA Extreme Computing Initiative [30], that are limited to operating on the costliest and massive supercomputers. These days, no one would construct an HPC cloud for these specific big science major challenges. The "HPC market" is simply too tiny to be viable, and as a result, there is no economies of scale. A hybrid infrastructure—cloud capacity resources coupled with HPC capability nodes—might make sense in some particular science application scenarios with complicated workflows made up of various tasks (workflow nodes). This would provide the best of both worlds.

2.8 Related Works

A lot of research and development has gone into the topics of rigorous job scheduling and efficient resource management on cluster systems. Numerous academics are looking for solutions for dynamic resource management and scheduling because of the adaptive nature of applications and their increasing complexity. However, several have also noted that it can be difficult to provide support for jobs that are changing [40] [57]. The majority of the work in this field is theoretical or simulation-based. Dynamic scheduling based on evolutionary algorithms was carried out in one of the first efforts on scheduling evolving jobs [8]. The method was assessed using simulators. It was not addressed, though, if the method might be applied to more intricate scheduling elements like backfilling, fairsharing, and priority. After looking into the RMS needs for changing jobs, they [81] developed protocols to accommodate them and put in place a model RMS to examine the

overhead associated with dynamic allocation. The issue of organizing jobs that change over time was overlooked. Using the CooRMv2 RMS, the problem of scheduling jobs that change unexpectedly was examined [12]. According to their methodology, the maximum amount of resources that might be dynamically needed during execution must be specified at task submission, in addition to the standard work requirements. Only jobs that are preemptive or malleable may utilize these extra nodes, which have been preallocated for the job. Preemptive jobs are jobs with lower priority that have the ability to be canceled mid-run and rescheduled to free up nodes for jobs with higher priorities. The authors demonstrate the advantages of their method by evaluating it against a workload of changing and adaptable assignments. However, the preallocation method's usefulness is restricted to a specific class of changing occupations, and in the absence of jobs that are preemptive or pliable, it may result in a sharp decline in performance Additionally, the Moab Workload management [Moab h] and the SLURM resource management [M. A] support dynamic allocation on demand. Moab periodically queries each application about its load, allowing it to accommodate resource growth and shrinkage for changing jobs. This is limited to interactive workloads, though. By enabling a running job to submit a new job with a dependency indicator and then merging the allocations, the SLURM resource manager provides expand/shrink operations. The dynamic request is prioritized using the current static fairshare process by submitting a new job. But in this study, new scheduling fairness approaches are introduced and dynamic and static requests are distinguished. Furthermore, during a dynamic deallocation, all the resources allocated for a changing request must be released simultaneously per SLURM's design. Our method does not impose any such constraint, therefore offering greater flexibility. Jobs have the ability to deallocate any portion of their existing allocation dynamically. From an alternative perspective, [D. Kumar] solved the fairness issue by pointing out that jobs increase walltime allotment instead of using additional resources. Their method relies on adding features to a look ahead optimizing scheduler (LOS), which determines the optimal set of tasks to execute concurrently while maximizing resource usage. It is not practical to use the method for dynamic allocations, though.

A few years ago, it was theoretically determined that changeable vocations provide advantages. Because of this, frameworks for creating flexible applications have been around for a long time [83] [41]. By utilizing an equipartitioning policy to allocate idle resources evenly among malleable jobs and an experimental scheduler,

they [53] constructed an adaptive runtime system for Charm++ and demonstrated the advantages of malleable jobs over rigid ones. Efficient scheduling and resource management for flexible occupations have been extensively researched ever since. The majority of research in this area focuses on theoretical issues related to scheduling and simulation-based evaluation. For instance, researchers [88] suggested an online scheduling approach for jobs that were changeable, with the primary objective being the assignment of resources to jobs like to assign resources to jobs such that the total running time of the application is reduced. When submitting a job, users indicate how long they believe the job will take to complete on a single processor. Users received incentives if their job was finished by the deadline in order to prevent them from manipulating the scheduler by reporting the job's parameters incorrectly. H. Sun suggested a scheduling method in which resources are equipartitioned to flexible jobs. The strategy is then periodically changed based on feedback from the applications regarding the jobs' scaling pattern. Similar strategies were employed by [39] and [25]. Notable prototype/demo schedulers for modifiable jobs are covered here. Their normal scheduling process involves the scheduler downsizing already extended malleable jobs in an attempt to obtain nodes for the job when resources become unavailable and the queued job with the next highest priority cannot be launched. An expansion phase is initiated when the next queued task, even with smaller workloads, is unable to consume any resources. In this phase, idle resources are shared among the running malleable jobs in order to increase throughput and system utilization. The policy determines the different sequence in which jobs are chosen for expansion and shrinkage. [44] demonstrated how equipartitioning shrink and expand projects can greatly speed up reaction times by using moldable and malleable materials. An FCFS-malleable technique, as proposed by the authors [27], assigns available nodes to malleable jobs in the order of earliest-started-job. They also looked into alternative approaches, such putting the least amount of CPU usage first, the earliest deadline first, and the latest deadline first. They demonstrated that, in comparison to the well-known EASY backfilling, the earliest began first technique generally performed better and improved the average response time by 31% for a cluster consisting exclusively of flexible jobs. Support for flexible MPI jobs was also added to the OAR resource manager [57] [58]. The issue of arranging several flexible jobs, however, was not covered. Within the framework of grids, their suggested solution [42] added malleability to the KOALA multicluster grid scheduler, offering

an alternative take on equipartitioning in the form of an equigrow and equiishrink strategy. The equigrow strategy just divides the available idle resources among the malleable tasks evenly, whereas the equipartition policy attempts to balance the number of malleable nodes owned by each active malleable job. Therefore, when the scheduler initiates an expansion phase, a work will be enlarged by an identical fraction of idle nodes, regardless of the number of nodes held malleably by the job. Precedence to running applications (PRA) and precedence to waiting applications (PWA) were two scheduling strategies that were combined with this. PRA prioritizes the allocation of idle nodes to running malleable jobs over queued jobs in an effort to increase the number of running malleable jobs. PWA, on the other hand, makes an effort to assign every job in line before taking into account the expansion of running jobs that are flexible.

Almost all batch systems in use today allow moldable work scheduling, as it is not as technically difficult to enable as it is to support developing and malleable jobs. Among them are Moab HPC Suite [62], OAR [57], SLURM, and Platform LSF [73]. Over the past 20 years, numerous approaches to moldable project scheduling have been put out, with the primary goal being an increase in average turnaround time. [89] suggested one of the earliest efforts on scheduling moldable jobs. In an effort to provide the fastest turnaround time feasible, they unveiled Supercomputer AppLeS (SA), a moldable application scheduler. It uses a resource allocation list and a submit-time greedy technique based on the current system state to simulate submissions made to the system and analyze each request, estimating the turnaround time for each operation. Processors are then assigned by the scheduler to tasks that are anticipated to complete the quickest. A fair share method was proposed by Srinivasan et al. [84] [86] when they noticed that an avaricious plan may result in an uneven allocation of resources for small jobs. Through extended reservation policies that restrict the widths of job partitions, this was further improved to be resilient under varying load and scalability situations [85]. An iterative processor allocation approach was proposed by Gerald et al. [26], wherein decisions are based on work efficiency calculated using the Downey model [1], overbooking, and fair share allocation. Additionally, moldable job scheduling has been investigated in a number of contexts, including cloud settings [50] and frameworks including schedulers and runtime for SAT solvers [80]. Recently, Wu et al. [87] proposed the HRF (highest revenue first) strategy, which

divides up processor allocations based on how much money can be made by cutting a job's runtime.

The majority of prior research on the integration of predictive tools with HPC schedulers has concentrated on making direct predictions about job queue times, or the interval between a job's submission and its commencement of execution. Once users have an idea of which system or resource request size (e.g., number of nodes) will yield the shortest turnaround time (i.e., the time between a job's submission and completion), these queue time forecasts are typically given to them. Downey created a statistical model that uses only the jobs that are currently queued or executing on an HPC system to forecast a job's queue time [3]. To offer wait time limitations for jobs with measurable confidence levels, they created the non-parametric QBETS approach [57]. QBETS computes a bound on the job start time based on related jobs it finds using clustering to identify past jobs that are comparable to the current job. Later, Nurmi et al. expanded QBETS to help users reserve jobs probabilistically without the target HPC system's assistance [89]. Qespera, an adaptive framework developed by Murali and Vadhiyar, predicts queue times based on past jobs as well as past system states that are comparable to the current job and state. Utilizing runtime predictions to enhance job backfilling has been the main focus of the remaining prior work on integrating predictive tools with HPC schedulers. In order to incorporate job runtimes anticipated by taking a simple average of the preceding two jobs, the suggested system modified a backfilling scheduler [60]. The work in question use the projected runtime for all other tasks, such as backfill constraint checks, but retains the user-provided walltime for the kill-time (i.e., the moment at which the system terminates the job). While Gaussier et al. construct a machine learning model tailored for the task, Tsafrir et al. apply the same basic technique [28]. In addition to being more accurate, the more complicated model enables adjustments to the predictions to favor various task types (e.g., short versus long-running) and error types (e.g., under- versus over prediction). To the best of our knowledge, our work is the first approach to use in a batch task scheduler predictions other than runtime.

Rather of ranking tasks based on the job itself, fairshare scheduling assigns a higher priority to task owners based on their historical resource usage. It comes from a technique used by the operating system's job scheduler [51]. This technique creates a "fair" model of resource sharing that allows users to acquire system capacity proportionate to quota allocations, regardless of the number of tasks they have

running on the system (i.e. preventing starving of users with fewer tasks). In local resource management (cluster scheduler) systems such as SLURM [93] and Maui [61], fairshare prioritization is applied to the job level, influencing the scheduling order for jobs according to past resource capacity utilization. Administrators can now configure the relative relevance of fairsharing in systems by setting weights. Fairshare is frequently managed as one schedule component out of many. For distributed computing systems such as compute grids, a model for decentralized fairshare scheduling based on hierarchical allocation policy distribution is given in [21], and a prototype realization and evaluation of the concept is described in [68]. By addressing an extended approach for decentralized prioritization in distributed computing in [69], it expands on this work. Here, we expand on this work by tackling the design and assessment of priority operators, which are mathematical functions that compute the discrepancy between individual user quota allocations and historical resource consumption.

The traditional batch job scheduling model is used by the production HPC job schedulers, including Slurm, Moab/TORQUE, PBS, and Cobalt [51]–[23]. In this model, users request a fixed amount of resources for a predetermined period of time, and the scheduler determines the best time and location to run each job based on job priority and system availability. Several works try to solve the scheduling of hybrid workloads on a single HPC system. The high responsiveness of on-demand work is a common goal of research on coscheduling rigid and on-demand applications. Preempting rigid jobs to make place for on-demand employment, anticipating demands for on-demand jobs, and saving resources for on-demand jobs are frequent tactics [69], [21]. On HPC systems, co-scheduling flexible and stiff tasks is the subject of several studies [69]–[48]. Regretfully, the issue of co-scheduling all three categories of applications—that is, flexible, inflexible, and on-demand jobs—has not been addressed by any of the research that have already been done. Therefore, it is unknown what effects co-running these programs will have on scheduling. One HPC system running hybrid workloads may be able to accommodate ever-larger on-demand job sizes, lessen resource fragmentation, and increase system utilization through adaptable job types. Nonetheless, this is a difficult assignment in terms of resource management as well as job scheduling. A number of competing goals must be carefully balanced by the work scheduler, including low impact on rigid jobs, high system utilization, quick response to on-demand jobs, and an incentive for decreasing

malleable jobs. The resource manager is responsible for carrying out the task scheduler's more intricate and frequent activities, such as start, preemption, shrink, and expansion procedures. This calls for corporations with specialized applications and more effective drainage procedures.

2.9 Summary

This chapter described current trending technologies utility computing, cluster computing, grid computing, cloud computing, HPC and fog computing. HPC is useful for huge amount of data processing. So researchers are trying to research to improve latest technologies and scheduling algorithms.

CHAPTER 3
BACKGROUND THEORY

This chapter talked about the natures and concepts of high performance computing, as well as the similarities and differences amongst them. High Performance Computing (HPC) was defined by the Joint Information Systems Committee of the UK's New Technology Initiative using the following terms [2]: "Computing resources that offer a computational capacity greater than ten times that of a desktop computer." The term "high performance computing," or HPC, often refers to the process of combining a cluster of computers' resources to solve complicated problems that are beyond the capabilities of a single node. By developing high-end computation, storage, and networking, HPC delivers high level performance in comparison to general-purpose computing. HPC has never been more vital to the advancement of science for the good of humanity, from predicting earthquakes and hurricanes to solving the worldwide food crises.

3.1 Evolution of High Performance Computing

The evolutions of traditional HPC [37] for present and future systems are shown in Figure 3.1. The idea of high performance computing, or HPC, is not new to our sector or the wider world. Actually, it dates back to the 1940s, when the world's first supercomputer was constructed, combining millions of processors to create a potent computing system. From the earliest days of electronic mail and music streaming to the latest developments in weather forecasting and ChatGPT, the supercomputer has been at the center of technological innovation for decades. It was also the forerunner of HPC as we know it today, enabling new innovations in data processing, analysis, and storage. a modestly sized resource for a certain subject of study that only a few users would use. It was also somewhat manual. The parts weren't made especially for the purposes for which they were being utilized. At that time, there were just no such things as high-performance interconnects based on open standards, like InfiniBand, or GPUs for accelerated computing.

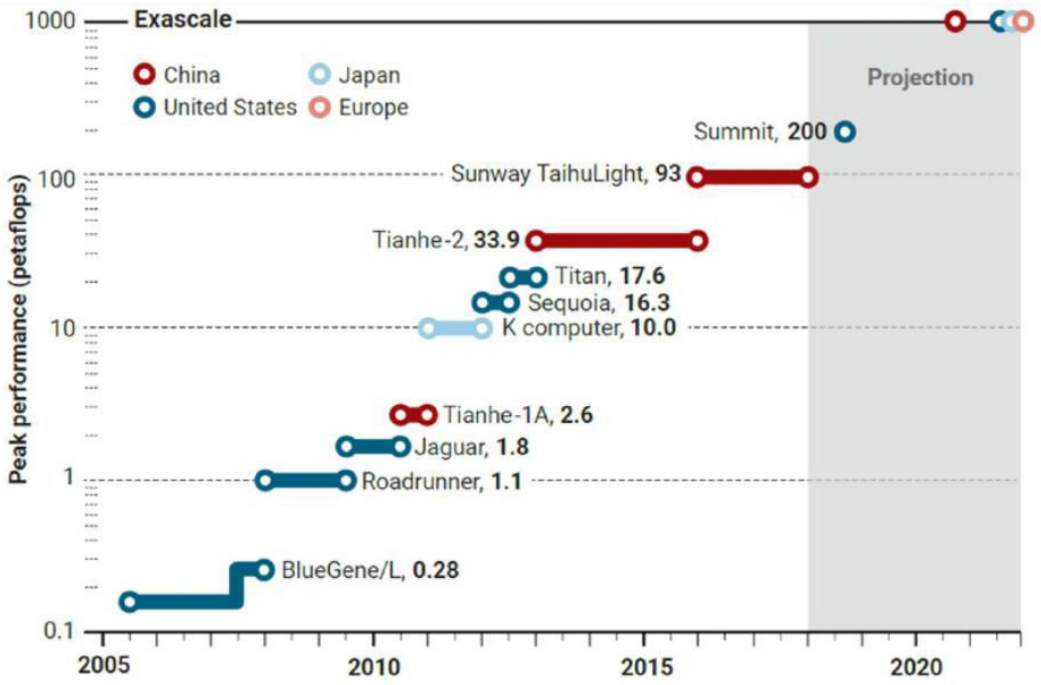


Figure 3.1 Timeline of HPC Evolution

Ever since data-driven goods began to appear, with the goal of creating a whole automated and intelligently linked world, best-quality computing resources are seen as essential strategic elements for breakthroughs and growth.

Cloud-based servers, specialized data centers, and HPC systems give small and medium-sized businesses and the industry the computational capability they need to achieve the goals of a smarter, more prosperous future for all people. Gaining more performance out of computer resources has been a top priority for application developers and hardware architects for many years. According to the implications of Moore's law [9], a dense integrated circuits transistor count doubles every 18 to 24 months. Dennard scaling [47] is a Moore's law postulation that states that transistor power consumption is precisely related to area. For many years, researchers have used Dennard's scaling and Moore's law as a guide to increase processor performance exponentially while maintaining a negligible increase in power consumption. Unfortunately, the discontinuation of Dennard's scaling about 2006 permanently halted the acquisition of additional benefits and performance enhancements. Consequently, the single-core, multi-core, and many-core platform structures were evolved by the processor makers. To prevent this kind of performance scaling, Moore's law's loss has implications for daily life [32]. Before the mid-2000s,

computer users started to expect processor performance to double every 18 months, or more than three decades, due to Dennard scaling and Moore's law. Both power and clock rate increased swiftly. From an HPC standpoint, this was the era of both homogeneous and heterogeneous clusters of single-core processors. However, by the end of 2003, power density—a measure of heat dissipation that is the amount of power dissipated per unit area—and constantly rising power consumption were posing a challenge to system designers [59].

3.2 HPC System

Performance-efficiency trade-offs and application requirements inevitably affect HPC systems. The features of these systems are discussed in this with reference to [NERSC's Edison], a Cray XC30 supercomputer that generates 2.5 Pflops/s of raw computing power. A supercomputer cannot function properly without a lot of software modules and libraries. Furthermore, the design of each supercomputer may determine which software packages are installed on it and to what extent.

User Application: A program that runs on the system and produces output that directly benefits the user is referred to as a user application. A weather simulation that forecasts the daily variations in climate is one example.

Programming models and systems: The most important software package required to run a parallel application is an appropriate parallel programming model and programming system, or paradigm. Depending on the application's architecture and logic, the user must choose a suitable model and paradigm. Message passing between many processes running an application in parallel has been the most often used mechanism in distributed memory systems. In this regard, the defector standard for message forwarding in high-performance computing is MPI [49] [67]. MPI processes can interact with one another in a number of ways, such as point-to-point and collective communication, and they normally run concurrently on every cluster node. An MPI program's process is a part of a communicator, which is an abstract environment. Each communicator has a unique identity known as a rank. On modern HPC systems, MPI is one of the most used parallel programming models.

The main method for achieving parallelism in shared-memory systems is multithreading. One well-known example of a library that facilitates multithreading in an efficient way is the POSIX threading interface (Pthreads). For HPC applications,

the OpenMP API [67] is the preferred option. Programmers can access an intuitive and adaptable interface with OpenMP according to its scalable and portable paradigm. Today's significantly expanding multicore systems have greatly increased the importance of OpenMP. Additionally, users utilize hybrid programming (MPI and OpenMP) to take advantage of parallelism at all levels. MPI provides cluster-wide parallelism, whereas OpenMP provides node-level parallelism.

The Charm++ parallel programming system [53] [64] [65] [72] [46] is a programming paradigm that works with both architectures. Scientists can run larger models at quicker speeds and finer resolutions due to HPC. Consider the field of study on earthquakes. Earthquakes of great magnitude wreak damage on human civilization. According to the significant developments in high-performance computing, scientists are able to push the boundaries of research aimed at minimizing seismic damage by simulating the intricate processes connected to large-scale earthquakes. For example, a 2017 effort [19] uses Sunway TaihuLight, the fastest supercomputer in the world at the time, to execute large-scale nonlinear seismic simulation. Their work achieves over 15% of the supercomputer's top performance, with extreme instances displaying a continuous performance of over 18.9 Pflops. This enables the simulation of the Tangshan earthquake as an 18-Hz scenario with an 8-meter resolution.

The growing use of HPC technology is not just confined to scientific computing; it also promotes innovation in new fields like Big Data, Internet of Things (IoT), and Machine Learning (ML). AeroFarms is a remarkable instance of a vertical farm powered by IoT and machine learning that regulates the growth and operation of plants at an incredibly precise level. The vertical farm collects information on all variables, including light, oxygen, and nutrients and moisture, and transmits it to high-performance computing centers that are tailored for machine learning. Complex decision-making is made possible by HPC technology, including real-time quality control that depends on a variety of data sources. The farm is able to use up to 95% less water than traditional field farming and greatly increase annual yield thanks to the deep integration of IoT, HPC, and ML [86].

The cluster scheduler must receive resource requests from users who wish to run applications, such as a series of tasks. In essence, the request identifies the (a) processor count and the (b) expected runtime required to complete the jobs. In response to the resource request, the scheduler gives resources temporary ownership, which occurs between a precisely defined start time and end time. In an HPC resource

scheduler, a job is the fundamental scheduling unit that defines this kind of temporary ownership. A job is different from an application in that the former might be thought of as a lease that is required by the latter. One straightforward technique to achieve high performance of an application from the perspective of resource efficiency is to grant the job corresponding to the application exclusive ownership of resources for a continuous length of time. This is a technique called "space share," as opposed to "time share" or "multi-tenancy," which permits the simultaneous use of several applications in a shared environment. Another key tenet of space sharing is assigning an independent processor to each task in an application so that space sharing is achieved at the task/processor level. To arbitrate space share, a job queue is managed by a batch scheduler. It controls the order in which tasks start running according to scheduling policies and available resources. A task can only begin when all of its resources are allotted in the processor's "shape" of time, allowing the application to operate continuously until it is finished or the job runtime ends. This work structure can be described as "all-or-nothing" and "rigid-job." A rigid job is one whose duration and size are set in stone. The scheduler's all-or-nothing conditions state that resources must be allocated entirely at once; otherwise, the job will have to wait.

Last but not least, the ability of a new generation of scientific apparatuses and devices—from sophisticated light sources to geographic information systems (GIS)—to generate massive amounts of experimental data that require rapid data analysis is a significant development in the field of high-performance computing (HPC). These new applications are more time-sensitive than traditional batch programs, which means that they can't wait around for lengthy wait times; instead, they need on-demand resource availability for data analytics. Additionally, quick data processing turnaround times allow for quicker scientific experiment iterations, which eventually quickens the pace of scientific discovery.

Many HPC system operators are developing on-demand service frameworks that facilitate rapid resource access in order to cater to this new class of applications. Table 3.1 described the components of the HPC system.

Table 3.1 HPC System

Term	Definition
Cluster	A high-performance computer made up of several machines that work remotely from the head node or login.
Master Node	When you join to a cluster, the computer you log in to is known as the master node. Software compilation and task submission are done via this node.
Storage Node	Cluster home directories are connected to a storage node, which is thus reachable from any location in the cluster, to lessen the strain on the master node. As a result, compute processes can communicate with the storage node directly while running, saving the master node's resources.
CPU Compute Node	On the CPU compute nodes, the CPU jobs are carried out. Jobs are distributed among CPU compute nodes via the job scheduler, which gets them from the master node. There are 77 CPU computing nodes in the Vikram-100 HPC.
GPU Compute Node	The GPU jobs are executed on the GPU computing nodes. The job scheduler receives jobs from the master node and distributes them to GPU compute nodes. The Vikram-100 HPC has 20 GPU compute nodes.
CPU Core	Every cluster node is equipped with two 12-core CPUs. A single node can run up to 24 jobs concurrently since each core can only handle one work at a time.
MPI	A technique called Message Passing Infrastructure (MPI) allows computing jobs to be executed on multiple nodes. Designed to operate in scenarios where discrete portions of the project can be completed on separate nodes, with the results being sent to other nodes to complete the remaining portions of the job.
Module	One way to allow access to several software versions without running the danger of version conflicts is to use the module command.
Job Script	A file that has all the necessary information to execute a job, including the commands to run the job and the list of resources required, is called a job script.
Job Scheduler	The job scheduler monitors the jobs that are running on the cluster and distributes open jobs among nodes based on the nodes the submitter can access, the resources needed for the job, and the cluster's recent usage. To sum up, a job scheduler determines the time and place at which a job is to be completed. We make use of an LSF task scheduler.
Interactive Job	To test your software and data on a cluster, use an interactive job. On one of the nodes, you request a terminal, after which you load and execute files via the command line. You should convert your job to a batch job after it is functioning successfully in an interactive manner.
Batch Job	A batch job is a cluster job that runs automatically after it is submitted and doesn't need any more input. Almost all of the jobs on the cluster are batch jobs.

3.3 Job Scheduling

The scheduler is in charge of choosing and allocating resources on the cluster for jobs that users submit. These resources are chosen by HPC schedulers in order to maximize goals such as scientific productivity and resource efficiency. A scheduler needs to take into account more factors than just these scheduling goals. The limitations imposed by system administrators must also be taken into account by schedulers. For instance, in contrast to cloud computing, where several applications from various users share the same resources, HPC systems often grant workloads exclusive access to those resources. Due to this exclusive access requirement, the scheduler frequently has to add a job to a queue and execute it later, once enough resources have become idle, in cases when there are not enough resources available to run a freshly submitted job [52]. The scheduling policy is the algorithm that decides how to handle the jobs that are queued.

The resource manager receives the job allocation information from the scheduler for execution once the scheduler has decided which jobs to perform and how many resources to assign to each job. The scheduler will be notified by the resource manager when the job is finished and if any resources become available or unavailable. These notifications are used by the scheduler to update its internal system model. When a job ends, resources should be swiftly redistributed and offline resources should not be assigned to new projects, according to an accurate internal model. Many schedulers maintain databases to hold past work and resource information in addition to an internal model of the current state of the system. For management, auditing, and enforcing equitable usage, this database is helpful.

3.3.1 Job Types

Based on the flexibility of resource consumption, parallel jobs can be divided into four groups.

(i) Rigid jobs: The most prevalent kind of work in today's cluster environments is a rigid job. A batch system receives a strict job that specifies the number of nodes required to run the parallel program. while a job is submitted, it cannot be changed while it is underway or during its execution. The method employed in the program itself, which can only be run on a set number of nodes, is the most common cause of

the rigidity found in the majority of parallel applications. For instance, an application's decomposition for a particular problem size might only be appropriate for a certain number of nodes. A strict job also establishes a time limit for the application's execution that it cannot exceed. Best-fit algorithms and backfilling are two of the many techniques available for scheduling inflexible operations with the goal of increasing throughput and response times. As long as the backfilling jobs don't cause a delay in the start time of a configurable number of jobs positioned higher up in the queue, backfilling is the process of scheduling jobs out of order from a FIFO queue.

(ii) Moldable jobs: Similar to a rigid task, a moldable job allows the batch system to modify its resource requirements after the job is submitted but before it begins. In order to optimize performance, the batch system may decide to map the job onto idle nodes that are now available, for example, or it may decide to execute the work with more or fewer resources than the primary resource need indicated. Consequently, a range of acceptable nodes and a certain execution time are specified for each specification when moldable jobs are submitted. As long as the application can divide the problem into manageable chunks based on the resources available for execution, MPI jobs are moldable.

(iii) Evolving jobs: An evolving job has the ability to modify its resource allocation set (or reservation) while it is being executed, in contrast to inflexible and moldable jobs. A developing job dynamically requests more nodes from the batch system, hence starting a shift in resource allocation. The batch system extends the task in response to this request. The term "job shrink" refers to the application's ability to simultaneously dynamically release part of its nodes and minimize the size of its reservation. A job may change for a variety of reasons. For instance, the application might have needed more computations to complete the task inside the allotted walltime limit because of intermediate results, which call for more resources. Applications might occasionally run out of resources before they can finish running (for example, when they hit memory or other hardware restrictions). In these kinds of situations, the program needs more resources in order to continue running and spread the data and calculations among them.

(iv) Malleable jobs: The best positions for schedulers are those that are malleable. A malleable project can be expanded or contracted by the batch system at any time. The program will adjust to the new resource set on its own. OpenMP is the most widely

used programming model that supports malleability. The parallelism between the program's parallel parts can be changed transparently as more of a node's cores become accessible to a process. It is useless, though, because production clusters dedicate all of their nodes to a single task. Multi-node malleability is possible with other adaptive programming paradigms such as Charm++, AMPI [11], and OmpSS [3]. The resource needs for a flexible job can be described in a number of ways. The most popular approach is defining a minimum, an optimal, and a maximum number of nodes needed for the job, allowing the allocation of the job to be increased or decreased within the given range. A flexible job's resource set can be dynamically changed by the batch system to maximize system throughput and usage. The user gains from this in a number of ways as well. For instance, a malleable work can be expanded later when more nodes become available, or it can be launched as soon as the minimal number of nodes needed are ready.

(v) On-demand jobs: An on-demand work is a time-sensitive application that must be finished as quickly as feasible. Workloads involving data analysis following experiments are an example of an on-demand employment [5]. Traditionally, on-demand operations run on dedicated clusters to provide high responsiveness. Small clusters cannot support the ever-increasing computational demands of the rapid proliferation of experiments. Large-scale HPC systems can be used as a practical answer to the constantly growing on-demand workloads.

3.3.2 Scheduling Metrics Scheduling Policies

These metrics are used to measure the scheduling performance (system performance).

3.3.2.1 System Throughput

The most crucial indicator of a batch scheduler's efficacy is system throughput. The number of tasks finished in a given amount of time is its definition. The throughput should ideally be as high as possible because it improves system availability. A scheduler's responsibility is to guarantee that jobs are executed effectively in order to maintain a good throughput and prevent jobs from experiencing resource hunger, as a cluster usually receives a mix of long and short running activities. usage of the system. System utilization is a measurement of how many

resources are being used by jobs, as the name suggests. By periodically monitoring resource utilization, it is typically expressed as a percentage of total resources consumed over a given period of time. To prevent wasting resources, the scheduler should strive to maintain a high system utilization rate. In addition to resulting in low throughput, resource waste raises system providers' expenses. Conversely, high throughput does not automatically equate to high system utilization. Without providing the highest throughput, an inefficient job and resource mapping might still result in excessive system utilization. Consequently, an effective scheduler needs to strive for both high throughput and high system utilization.

3.3.2.2 Makespan

Makespan is the amount of time needed to finish a set amount of work. A popular and practical statistic for evaluating and contrasting various scheduling methods is makespan. Better scheduling is implied by a lower makespan number, which also indicates higher throughput. However, because production machines don't have set workloads and frequently receive jobs at unpredictable intervals, it is inappropriate to use it to determine the quality of scheduling in a machine.

3.3.2.3 Waiting Time

The amount of time a job sits in the queue before being completed is known as the waiting time. Put another way, it's the interval between the start time of the job and the batch system submission time. The average waiting time can be used to assess a scheduler's efficacy for a particular workload. Better scheduling is indicated by a lower average waiting time. The best metric to assess fairness strategies is the waiting time. Generally speaking, the goal of a fairness policy that gives each user equal priority is to keep the average waiting time for jobs between users equal.

3.3.2.4 Turnaround Time

The entire amount of time that elapses between the job's submission and completion is known as the turnaround time. To put it another way, it's the sum of the work's execution time and its waiting time. When assessing schedulers, the average turnaround time is frequently employed (e.g., for a person, a job, a group of users).

The batch system can only affect a job's turnaround time via decreasing waiting times. The hardware and program dictate how long an execution takes. Nevertheless, the batch system may also affect the execution time of other kinds of activities. It is generally faster to provide more nodes or cores than what a moldable job requests. The execution time of an application may be decreased by expanding malleable jobs. Comparably, by granting their demands for dynamic resource allocation, evolving jobs can also be helped by the batch system to finish on schedule or even sooner. Thus, it is possible to assess static and dynamic allocation strategies using job turnaround time.

3.3.2.5 Reaction Time

The formal definition of reaction time is the amount of time that passes between the batch system's initial response and the time the job was submitted. Given that a work in the batch system is only produced once it is completed, this corresponds to the turnaround time for batch jobs as well. Not only are interactive jobs completed within the allotted time frame. This metric has been used interchangeably with work turnaround time by academics because the majority of jobs carried out in HPC clusters are batch activities.

3.3.2.6 Response Time

The amount of time a process spends waiting to get the CPU when it is ready is known as response time. The terms turnaround time, reaction time, and throughput have been used interchangeably because this thesis exclusively addresses batch processes.

3.3.2.7 Fairness

Since fairness is more of a trait than a metric, it cannot be quantified by a single metric. Simultaneously, it remains the most crucial attribute that a batch system needs to establish, given the exponential growth in HPC cluster users. Comparing the typical wait times for jobs among various users is a widely used method of assessing it. Fair scheduling is typically indicated by an average waiting time that is roughly similar. However, the average turnaround time of each user's collection of jobs may

differ if they have an uneven mix of different work kinds. Reputable batch systems typically combine these metrics to guarantee user equity. Above all, since equity is a political matter as well, it is determined in accordance with the requirements of a website by giving administrators the ability to define different factors, such the priorities of particular users or user groups. In order to encourage users to develop more flexible apps rather than inflexible ones, it is imperative to enable some degree of fairness in expand/shrink processes. Equipartitioning is a passably effective tactic for facilitating equitable dynamic allocation and deallocations. Equipartitioning by itself, however, is unable to increase the overall system throughput or reaction times since it may conflict with the choice of the most flexible for expand/shrink operations.

3.3.2.8 Goodput

Although goodput is mostly utilized in computer networks, scheduling can also benefit from its application. The pace at which meaningful data flows over a link in a computer network, without retransmissions and protocol overhead, is known as goodput. When it comes to scheduling, goodput refers to the amount of useful computation completed at any given moment, less the time spent waiting on disk and network I/O. When a link becomes congested and retransmissions are frequent, goodput in networks diminishes. Similar to this, scheduling experiences a decline in goodput when jobs compete with one another for access to system resources that have become overallocated [S. Herbein]. This makes it clear that there is a trade-off between goodput and resource utilization. The scheduler may need to run numerous resource-intensive jobs concurrently in order to achieve high utilization, which would reduce goodput. On the other hand, in order to avoid overusing shared resources, the scheduler might need to postpone some resource-intensive jobs in order to increase goodput [P. Balaji].

3.3.3 Scheduling Policies

Any overall scheduling policy consists of three sub-policies: a reservation policy, an ordering policy, and a resource selection policy.

3.3.3.1 Ordering Policy

According to this policy, tasks in the queue are ranked in order of submission date, user priority, history usage, and resources sought. Fair share, largest first, and First-Come-First-Served (FCFS) are a few ordering policy examples. The FCFS policy prioritizes the oldest jobs in the queue by ordering them according to the time of submission. Jobs are arranged in a queue order by size of required resources under the largest first policy. The largest jobs are at the front of the queue. Jobs are ranked according to a variety of weighted criteria specific to each computing location under the fair sharing policy. Fair share scheduling is designed to maximize the image of fairness, which will vary depending on the computing site. A job's priority, for instance, is determined by the user's past usage and the percentage of system time they have promised; users who exceed this time are assigned a lower priority at Lawrence Livermore National Laboratory (LLNL) [B. Barney]. These principles are occasionally mixed. For instance, on certain clusters at Oak Ridge National Laboratory (ORNL) and Lawrence Livermore National Laboratory (LLNL), fair share is blended with largest first by giving larger jobs a higher priority [43] [7].

3.3.3.2 Resource Selection Policy

This sub-policy chooses which specific resources to reserve or allocate to a job after assessing if there are sufficient resources to meet the job's criteria. The simplest resource selection strategy chooses the nodes to allocate arbitrarily, taking into account simply the number of compute nodes needed by the job. Nodes are chosen according to a network topology model in more intricate, network-aware selection procedures, which maximize locality [4] [64] [65]. A resource selection policy that considers input/output and removes contention from the parallel file system.

3.3.3.3 Reservation Policy

To decide whether to reserve resources for a particular job or allocate them immediately, this policy combines the ordering and resource selection policies. Queued jobs are taken into account by the reservation policy in the sequence specified by the ordering policy. The reservation policy uses the resource selection policy for each job in the queue to decide whether a reservation or allocation is required. The reservation policy will allocate resources right away if the resource selection policy finds that there are sufficient resources available right now; if not, it will create a

reservation for a later time. First come, first served (FCFS), cautious backfilling, and EASY backfilling are a few examples of reservation policies. The first job for which resources are not immediately available is when scheduling ends under the FCFS scheme (i.e., just allocations are created, no reservations are made). FCFS preserves job order and fairness exactly, but at a significant cost to resource utilization. The majority of the system will have to idle before the job is scheduled if there is a big job at the front of the queue. Backfilling can help prevent head-of-line blockage, as it is generally called. Backfilling allows jobs that are not currently able to be executed to have their resources provisionally reserved. Jobs that come later in the queue and do not conflict with any of the reservations will be scheduled. Every job is reserved in conservative backfilling, according to the sequence in which it shows up in the queue. Put otherwise, if filling the job would cause an idle position in any of the jobs that came before it, it cannot be backfilled. This significantly increases resource use without compromising fairness. A reservation is only made for the first job in the queue, or the first job that requires it, when using EASY backfilling. Put differently, backfilling is acceptable as long as it doesn't affect the first job in line. This further increases resource usage at the expense of diminished equity. There are hybrids that fall somewhere between conservative and EASY backfilling, with reservations held for a limited number of jobs that are larger than one.

3.4 Scheduling Algorithms

Algorithms for job scheduling are essential for computing systems to use resources efficiently. When it comes to controlling the distribution of jobs or processes across computing resources like CPUs, GPUs, or distributed systems, these methods are essential. Optimizing system performance measures including throughput, latency, fairness, and resource usage is the main goal of job scheduling algorithms.

3.4.1 First Come First Served (FCFS)

One of the most basic scheduling algorithms is FCFS, which executes jobs in the order they come in. Although FCFS is simple to use, it doesn't prioritize jobs based on their characteristics, which might result in poor average waiting times, particularly for long-running jobs [63] [90] [45]. First-come- first-served (FCFS) scheduling is the simplest scheduling method, however it might cause very long

processes to wait for very short processes. The FCFS algorithm is a nonpreemptive one.

3.4.2 Shortest Job First (SJF)

SJF scheduling selects the job as the smallest execution time next. This algorithm aims to reduce average waiting time and is optimal in terms of minimizing average turnaround time. However, it requires knowledge of job execution times, which might not be available in real-time systems [63]. Provably, the best scheduling strategy is shortest-job-first (SJF), which results in the shortest average waiting time. But it's challenging to implement SJF scheduling since it's hard to predict how long the next CPU burst will last. The general priority scheduling algorithm, which merely assigns the CPU to the process with the highest priority, is a specific instance of the SJF algorithm. Starvation may affect SJF scheduling as well as priority scheduling. One strategy to avoid starving is to age.

3.4.3 Round Robin (RR)

RR is a preemptive scheduling algorithm where each process is assigned a fixed time slice (quantum) for execution. After a job consumes its time slice, it is set back in the ready queue, and the scheduler selects the next job in line. RR provides fair allocation of CPU time among processes but may suffer from poor performance for tasks with varying execution times [90]. For a time-shared (interactive) system, round-robin (RR) scheduling is more suitable. For q time units—where q is the time quantum—RR scheduling puts the CPU to the first process in the ready queue. If the process hasn't given up the CPU after q time units, it gets preempted and transferred to the back of the ready queue. The choice of the time quantum is the main issue. When the quantum is too tiny, scheduling cost in the form of context-switch time becomes excessive; if the quantum is too high, RR scheduling degenerates to FCFS scheduling.

3.4.4 Priority Scheduling

Priority scheduling assigns priorities to jobs principled on factors such as job type, importance, or user-defined criteria. The scheduler chooses the job with the highest priority for execution. While effective for real-time systems and prioritizing

critical tasks, it can lead to starvation of low-priority jobs if not implemented carefully [90].

3.4.5 Multilevel Queue Scheduling

In multilevel queue scheduling, jobs are divided into separate queues based on different criteria such as priority, job type, or resource requirements. Each queue may have its scheduling algorithm, allowing for different scheduling policies for different types of jobs. This approach provides flexibility and can handle a diverse range of workloads efficiently.

3.4.6 Multilevel Feedback Queue Scheduling

The ability for processes to switch between queues in response to their actions makes multilayer feedback queue scheduling an extension of multilevel queue scheduling. I/O-bound jobs may be promoted to higher-priority queues, whereas jobs that consume excessive CPU time may be moved to lower-priority queues. With mixed workloads, this dynamic adjustment aids in maximizing system throughput and responsiveness.

3.4.7 Fair Share Scheduling

Fair share scheduling ensures that each user or group receives a fair share of system resources over time. It prevents any single user or group from monopolizing resources and allows for equitable resource allocation in shared computing environments.

3.4.8 Min-Min Scheduling

The current cloud scheduling technique is still based on the straightforward Min-Min algorithm [71]. An initial set S of all unmapped jobs is used. Next, it is possible to identify resource R, which has the lowest job completion time overall. The task T with the smallest size is then chosen and allocated to the relevant resource R,

hence the term Min-Min. Finally, Min-Min repeats the same process until all tasks are allocated (i.e., set S is empty) after removing task T from set S.

3.4.9 EASY Backfilling Scheduling

In parallel computing settings, a scheduling approach called EASY backfilling is utilized to maximize resource efficiency. As long as doing so doesn't cause the larger jobs' completion to be delayed, it permits smaller jobs to take up schedule gaps before larger jobs.

3.5 Simulation

The simulated application behaves more like real-world systems thanks to platform and application traces. Traces from artificial or real-world systems are both possible. Based on statistical analysis of real-world systems, synthetic traces capture variances that might not be obvious on real traces. In HPC, job scheduling is in control of assigning jobs to resources based on scheduling algorithms and resource availability, as well as organizing jobs in a waiting list. Real-world or artificial data are used to represent simulation data. The statistical study of actual systems serves as the foundation for synthetic data. Variations that might not be seen on actual traces can be captured by it. For the purpose of predicting and analyzing the performance of both present and future HPC systems [74] [6], modeling and simulation are crucial. Simulators give customers a quick and dependable way to assess programs that run on a certain platform. The variety of simulators has grown over time, giving rise to both general-purpose and specialty simulators on topics such as workload planning, resource allocation, energy efficiency, network modeling, network-aware scheduling, service brokering, and storage modeling [22] [78] [38] [56] [14] [78] [13]. GridSim [79] enables the simulation of basic tasks, network structure, data storage, and other helpful capabilities. It also offers simple implementation of common entities, such as computation resources or users. To meet increasingly complicated requirements, these entities had to be extended because the implementations that were given were too simplistic. This was accomplished by creating a new Java class that is an inheritor of the GridSim class that already exists. The intended functionality of this new entity can then be provided by adding or changing new parameters and functions. Because every crucial element, such as the job and Grid resource, is described in a different class, it's quite simple to edit or add new ones.

- **Accasim** : because of its minimal memory footprint and simple installation, Acsim is an open source, freely available Python library that can be used with any major operating system and is executable on a number of platforms. Because AccaSim can scale to large workload datasets and supports easy customization, experiments can be carried out using a variety of workload sources, resource types, and dispatching strategies. [10] [33].
- Batsim is provided the groundwork for the simulation framework SimGrid [29]. Although Batsim decouples the dispatcher from the simulator and makes it possible to implement it in any programming language, the dispatcher's source code and binaries are only available for GNU/Linux [34].
- **Alea** is built upon the GridSim simulation platform [78] The predetermined workload structure, resource categories, and dispatchers are what drive job submission, resource management, and job dispatching. The Java implementation is cross-platform and open-source. [15] [16] as well as [17].

3.5 Summary

Within this section, more than ten times the power of a conventional desktop computer is what is meant to be classified as high performance computing (HPC). In high-performance computing (HPC), numerous computers are combined to solve complex tasks. Through the integration of networking, storage, and high-end computing, this aggregate of computing resources improves performance. HPC is now essential for scientific progress, helping with everything from predicting natural disasters to solving global issues. The demand for higher performance has fueled HPC's development. Historically, increasing clock speeds without appreciably increasing power consumption was made possible by Dennard scaling and Moore's Law, which allowed for exponential gains in processor performance. However, in 2006, these principles' shortcomings prompted the creation of multi- and many-core CPU designs. In general, HPC is essential to enabling complex simulations, advancing scientific research, and driving innovation across various fields.

CHAPTER 4
THE PROPOSED SYSTEM ARCHITECTURE

The World Wide Web is quickly becoming the standard for commercial data exchange thanks to HPC [14]. High performance computing (HPC) devices are used for simulations and calculations that enable several scientific and industry research achievements. These systems usually comprise of high bandwidth concurrent file systems connected by low latency synchronous networks, and uniform, mostly parallel computing resources. Batch schedulers manage HPC systems, arranging application job execution to optimize utilization and control turnaround time. In the past, more powerful systems with more computing nodes, better transistor densities, and faster processor operating frequencies were built to meet capacity demands. Unfortunately, the limitations of semiconductor technology limit the potential for additional increases in processor frequency. Instead, while the capacity of individual processing units stays constant, parallelism inside processors and among compute nodes is growing. To keep up with the processing capability of the systems, memory and I/O hierarchies in HPC systems are also getting deeper and more complicated. HPC applications are also evolving: data processing and data-intensive applications are becoming more and more important as a result of the requirement to evaluate massive data sets and simulation results. Furthermore, the way applications are composed within HPC centers through workflows is becoming more and more significant.

4.1 The Proposed System Architecture

Time-sharing and space-sharing are the two primary categories of scheduling algorithms. Time-sharing algorithms partition the processor's working time into multiple distinct times, or slots, and allocate these to designated tasks. On the other hand, space-sharing algorithms allocate the necessary resources to a specific task until it is executed completely. Usually, cluster schedulers operate in space-share mode.

Pseudo-Code for Min-min Algorithm

```
1:  for i = 1 to M        // M denotes the number of tasks to be scheduled
2:      for j = 1 to N    // N denotes the number of virtual machines
3:          C_ij = E_ij + R_j
                          // C_ij denotes the completion time of task
                          // E_ij denotes execution time of task
                          // R_j denotes the ready time of task i on virtual machine j
4:      end for
5:  end for
6:  do until  all the unscheduled tasks are exhausted
7:      for each unscheduled task
8:          find the minimum completion time of the task and virtual machine that obtains it
9:      end for
10:     find the task t_p with earliest completion time
11:     assign task t_p to the virtual machine that gives the minimum completion  time
12:     delete task t_p from pull of unscheduled tasks
13:     update the ready time of the machine that gives the minimum completion time
14: end do
```

Figure 4.1 Pseudo-Code for Min-Min Algorithm

The Experimental Min-min creates all unmapped jobs using the set M. Next, the collection of minimal completion times, N, for any ti ϵ M is found. First, we select the function and assign it the consistent unit and the full minimum completion time from N. The newly mapped task is ultimately separated from M when all tasks have been mapped together, as seen in figure 4.1. Next, the procedure is repeated using a minimum completion time (MCT) that is based on min-min. In each mapping choice, Min-min considers all unmapped tasks, whereas MCT considers only one job at a time. The Min-min could concurrently complete multiple lengthy jobs while completing one shorter task if there is one short task. In this instance, rapid task execution is used to calculate the total makepan. The incoming jobs in the suggested system are categorized according to their potential to shorten the Min-Min algorithm's execution time. Machine clusters in HPC systems are given the right tasks to do. The primary drawbacks of the Min-min algorithm are job hunger with extended service times and load imbalance. Workload files are read by the workload reader and are specified (Standard Workload Format: swf format). The scheduler receives workload from the workload generator. The suggested technique, called the scheduler, chooses a resource machine by scheduling it based on procedure and configuration system. And after scheduling, the result collector retrieves the result from the workload file. The graph will be displayed in the output in.csv or visualization format.

In the proposed system, log files as job to the scheduler. The scheduler will schedule according to the proposed algorithm (combination of Min-min and priority algorithms) and work on system. The Configuration system has the number of

resources of the system configurations such as PE numbers. The proposed system architecture involves several components working together efficiently allocate computational resources to various tasks or jobs.

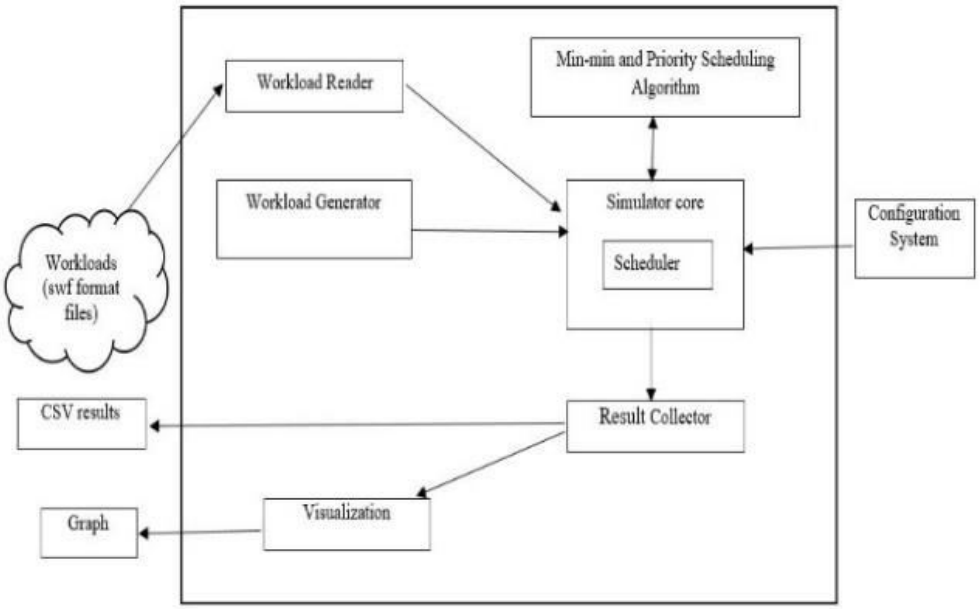


Figure 4.2 Proposed System Architecture

4.2 Proposed System (An Effective Job Scheduling Algorithm)

In Figure 4.3, when user submit batch job, calculate job size and expected execution time of job (job size= the number of cores the job requests). If job size is less than or equal available resources, choose job with minimum expected execution time of job. And then job size is less than or equal threshold value, these jobs are assigned to small compute node, in that time, small compute node is busy, assign to large compute node. Other side, job size is not less than or equal threshold value, these jobs are assigned to large compute node. When job size is not less than or equal available resources, wait until the resource are available. After allocating resources, update the job set and compute node list.

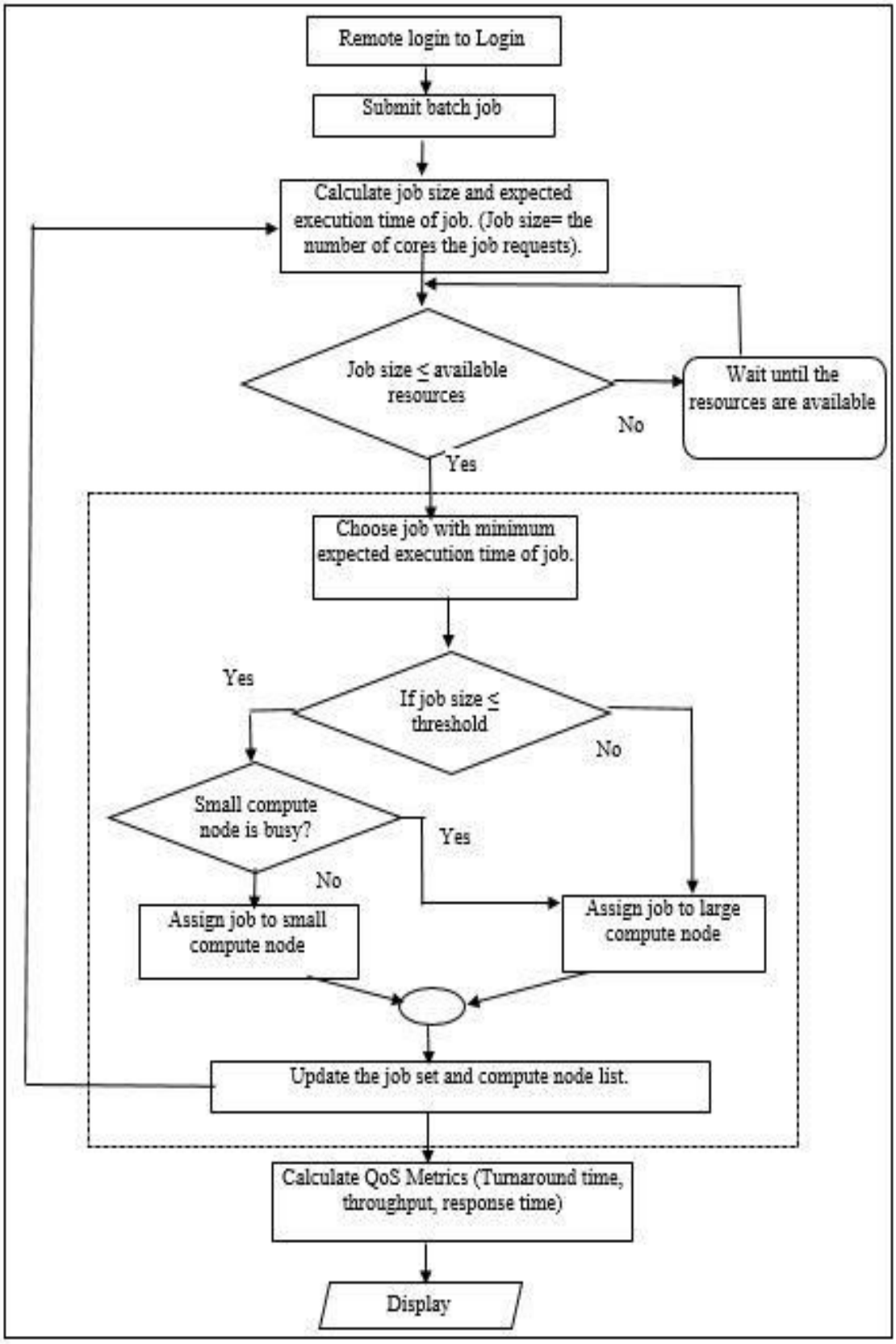


Figure 4.3 Flow Chart of The Proposed System

(1) For all submitted jobs t_i //in the set
(2) For all resources r_j
(3) $Ct_{ij}=Et_{ij}+rt_{ij}$;
End For ;
End For;
(4) Do while jobs set is not empty
(5) Find job t_k that cost minimum execution time that give minimum completion time.
(6) If job size (t_k) ≤ threshold value, assign t_k to small node r_j
If small node r_j is not available and then assign t_k to large node r_j
Else assign t_k to large node r_j
(7) Remove t_k from the jobs set
(8) Update ready time rt_j for select r_j
(9) Update Ct_{ij} for all t_i
(10) End Do

Figure 4.4 Proposed Job Scheduling Algorithm

Each job must be assigned to one or more resources by the scheduler in order for them to be completed. An algorithm for enhancing system performance has been devised for the suggested system. Let j represent the number of resources required to do i jobs. The set of all jobs in the suggested method is denoted by t = {t_1, t_2, t_3, …t_i}, and the set of all resources that will be mapped to jobs is denoted by r = {r_1, r_2, r_3, …… r_j}. The predicted completion time (Ctij) of each task ti on each resource rj is determined by the algorithm; each resource may have had a work allocated to it in the past. Ct_{ij} stands for completion time, Et_{ij} for expected task execution time on resource j^{th}, and r_j for the ready time for resource j^{th} (r_j is the ready time or availability time of resource j after completing previously assigned jobs).

The scheduler, which is the suggested system, is in charge of making schedules and doing additional optimization when clusters of resources and workloads are operating in an HPC system. The scheduler is managed dynamically when jobs come in during the system's simulation. The timeline is subject to change when new jobs arrive because some jobs are completed in this state. Compute the job size and execution time (in the simulator, these are already in the log files) when users submit batch jobs to the system. The job with the fastest anticipated execution time should be given priority if its size fits within the resources available. Give the job to a small computing node if its size falls below a predetermined threshold (processor count).

Assign the task to the large compute node if the small compute node is already in use. A large compute node should be assigned to handle the job directly if its size surpasses the threshold. Next, make updates to the compute node list and job sets. The results of the scheduling process are the QoS metrics (throughput, turnaround time, and response time) calculations.

4.3 Summary

This chapter mentions detailed explanation of the proposed system. The proposed system architecture incorporates a scheduling algorithm that combines Min-min and priority algorithms. Min-min scheduling assigns tasks based on minimum completion time (MCT), handling multiple tasks simultaneously to minimize total execution time. However, Min-min can lead to load imbalance and job starvation for tasks with long service times. In the proposed system, incoming tasks are classified to reduce execution time, and machine clusters are appropriately assigned tasks. The system involves components such as a workload reader, workload generator, scheduler, and result collector. The scheduler uses the proposed algorithm to allocate resources efficiently, and results are visualized or output in .csv format. Overall, the proposed architecture aims to improve resource allocation, reduce execution time, and address the limitations of existing scheduling algorithms in HPC systems.

CHAPTER 5
DESIGN AND IMPLEMENTATION OF THE PROPOSED SYSTEM

Nowadays, the usage of HPC systems are also increasing. The high performance computing (HPC) scheduling is responsible for assigning available resources to execution jobs in an effective way that increases system performance, decreases make-up and efficiently utilizes resources. Developing it is costly and time -consuming when researchers are expected to access the ready-to use testbed infrastructure for HPC is not available. If the testbed is accessible, it is restricted to a small number of resources and domains, testing the scalability and adaptability of scheduling algorithms, and assessing scheduler performance for different resource situations and applications is more difficult to track down. From the standpoint of the user, the system must provide quick response times, equitable resource distribution amongst jobs, and high system utilization and throughput. This research is presented the importance and the need for such infrastructure for environments of modeling and simulation.

5.1 Input The Proposed System

The proposed scheduling algorithm produces the effective performance than existing algorithms to generated measurement metrics. It uses the workload files from log files [93].

The workload files are shown in Figure 5.1. The original logs are available in several formats. To the extent that it is accessible, each log's unique format is described in the notes document that goes with it. All logs are converted to the Standard Workload Format (SWF) in addition to their original format. When available, it is advised to use the cleaned versions of certain of the logs [92].

= Important notes associated with log
= File to download
Mon = Duration of log in months
UE = Indicates whether log contains user runtime estimates
[You can add artificial estimates to logs that lack them by using this utility]
Mem = Indicates whether log contains data about memory (requested or used)
Util% = Utilization expressed as percentage

The Log Files:

#	Name	From	To	Mon	CPUs	U	M e m	Standard Format: *CLEANED* Jobs	Users	Util%	File	Standard Format: *ORIGINAL* Jobs	Users	Util%	File	Original Log
1	NASA iPSC	Oct 1993	Dec 1993	3	128			18,239	69	46.6	204K	42,264	69	46.7	400K	290K
2	LANL CM5	Oct 1994	Sep 1996	24	1,024	√	√	122,060	213	74.4	2.1M	201,387	213	75.2	2.8M	14M 3.6M
3	SDSC Par95	Dec 1994	Dec 1995	12	400			53,970	98	71.6	761K	76,872	98	71.7	1.1M	1.4M
4	SDSC Par96	Dec 1995	Dec 1996	12	400			32,135	60	75.6	487K	38,719	60	75.6	585K	831K
5	Early CTC SP2	Jun 1995	Apr 1996	10	430		√	*use original*				75,944	642	62.9	1.4M	
6	CTC SP2	Jun 1996	May 1997	11	338	√		77,222	679	85.2	1.5M	79,302	679	85.2	1.5M	3.5M
7	LLNL T3D	Jun 1996	Sep 1996	4	256			*use original*				22,779	153	60.1	243K	569K 246K
8	KTH SP2	Sep 1996	Aug 1997	11	100	√		28,476	214	70.1	402K	28,489	214	70.4	402K	722K
9	SDSC SP2	Apr 1998	Apr 2000	24	128	√		59,715	437	83.4	1.2M	73,496	437	83.7	1.4M	3.1M
10	LANL O2K	Nov 1999	Apr 2000	5	2,048		√	*use original*				122,233	337	69.7	2.1M	2.4M
11	OSC Cluster	Jan 2000	Nov 2001	22	178			36,097	253	12.8	524K	80,714	254	13.8	1.1M	2.0M
12	SDSC BLUE	Apr 2000	Jan 2003	32	1,152	√		243,306	468	76.7	3.9M	250,440	468	76.8	4.0M	6.0M
13	Sandia Ross	Nov 2001	Jan 2005	37	1,524	√	√	57,882	203	49.9	1.1M	85,355	204	50.2	1.5M	
14	HPC2N	Jul 2002	Jan 2006	42	240	√	√	202,871	257	60.1	2.9M	527,371	258	70.2	6.7M	13M
15	DAS2 fs0	Jan 2003	Jan 2004	12	144	√	√	*use original*				225,711	102	14.9	2.2M	
16	DAS2 fs1	Jan 2003	Dec 2003	12	64	√	√	*use original*				40,315	36	12.0	376K	
17	DAS2 fs2	Jan 2003	Dec 2003	12	64	√	√	*use original*				66,429	52	19.5	641K	14M
18	DAS2 fs3	Jan 2003	Dec 2003	12	64	√	√	*use original*				66,737	64	10.7	575K	
19	DAS2 fs4	Feb 2003	Dec 2003	11	64	√	√	*use original*				33,795	40	14.5	311K	
20	SDSC DataStar	Mar 2004	Apr 2005	13	1,664	√		96,069	460	67.2	1.6M	96,089	460	63.1	1.6M	
21	LPC EGEE	Aug 2004	May 2005	9	140	√	√	234,889	56	24.4	2.6M	244,821	57	20.8	2.7M	14M
22	LCG	Nov 2005	Dec 2005	1	24,515			*use original*				188,041	216		1.9M	1.3M
23	SHARCNET	Dec 2005	Jan 2007	13	6,828		√	*use original*				1,195,242	412	43.6	18M	31M
24	SHARCNET Whale	Jun 2006	Jan 2007	7	3,072		√	*use original*				589,251	154	72.1	8.6M	
25	LLNL uBGL	Nov 2006	Jun 2007	7	2,048	√		*use original*				112,611	62	56.1	732K	1.2M
26	LLNL Atlas	Nov 2006	Jun 2007	8	9,216	√		43,778	131	70.5	492K	60,332	132	64.1	633K	1.2M
27	LLNL Thunder	Jan 2007	Jun 2007	5	4,008	√		121,039	283	86.7	1.3M	128,662	283	87.9	1.4M	2.2M

Figure 5.1 Workload Log Files

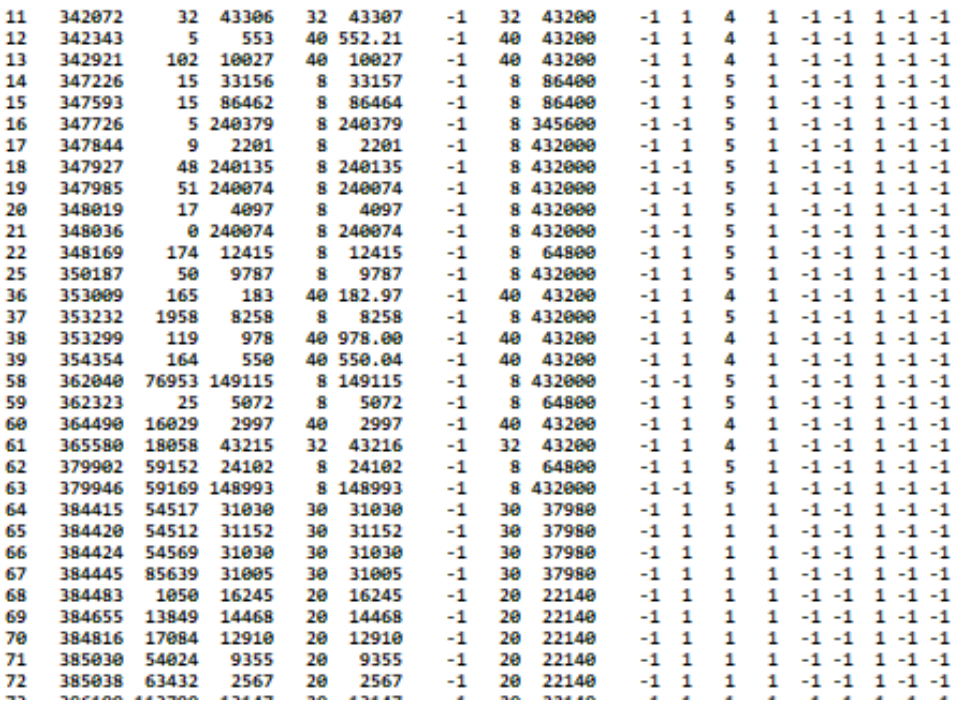
The data is presented "as is" in the ORIGINAL SWF files, with only small adjustments made to ensure consistency (e.g., jobs that appear to start before they arrive are adjusted to match the arrival and start times). Unfortunately, the data frequently contains undesirable and unrepresentative information, such as large-scale bursts of activity by a single person or extensive automated administrative activity. They offer a CLEANED version of the logs and advise using them in order to make using them in performance reviews easier [92].

Table 5.1 Data Fields in Data Sets

No.	Fields
1.	Job Number -- a counter field, starting from 1.
2.	Submit Time -- in seconds.
3.	Wait Time -- in seconds.
4.	Run Time -- in seconds.
5	Number of Allocated Processors -- an integer.
6	Average CPU Time Used -- both user and system, in seconds.
7.	Used Memory -- in kilobytes.
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)
10.	Requested Memory (again kilobytes per processor).
11.	Status 1 if the job was completed, 0 if it failed, and 5 if cancelled.
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
14.	Executable (Application) Number -- a natural number, between one and the number of different applications appearing in the workload.
15.	Queue Number -- a natural number, between one and the number of different queues in the system.
16.	Partition Number -- a natural number, between one and the number of different partitions in the systems.
17.	Preceding Job Number -- this is the number of a previous job in the workload
18.	Think Time from Preceding Job -- this is the number of seconds

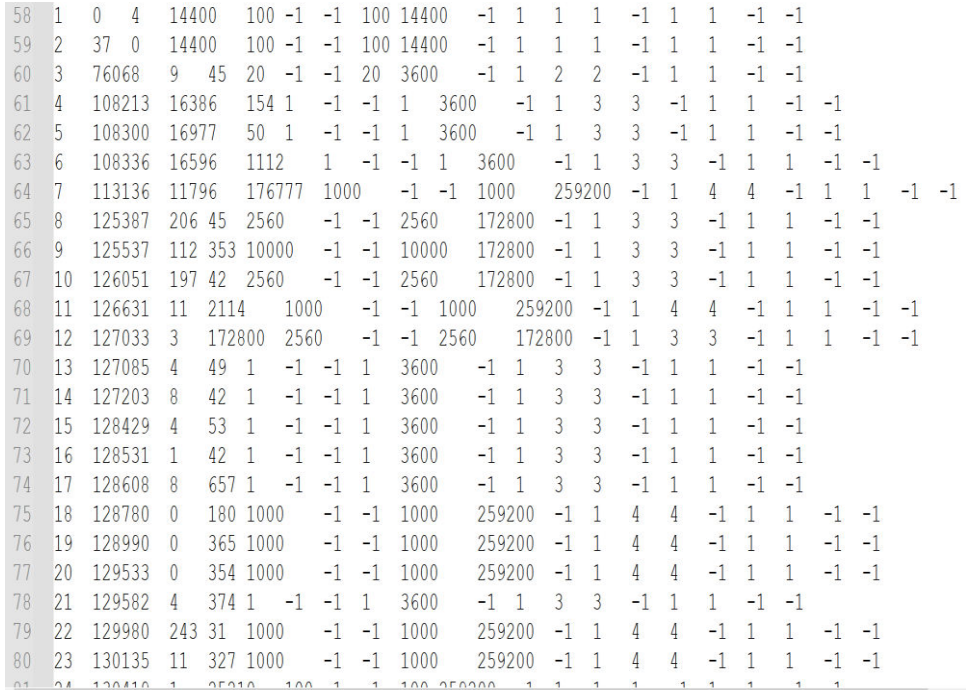
The format that was selected is an ASCII file with one line for each job, fields separated by spaces, and only numerical values allowed (no strings or unique date or time formats). Fields with no accessible data are indicated with −1.

The input workload file is shown in Figure 5.2, 5.3 and 5.4. In figure 5.2, HPC2N.swf workload file is showed. There are 527371 jobs are included. KIT-FH2-2016-1.swf workload file in figure 5.3, 114355 jobs are described. SDSC-SP2-1998-4.2-cln.swf workload file in figure 5.4 had 73496 jobs.



11	342072	32	43306	32	43307	-1	32	43200	-1	1	4	1	-1	-1	1	-1	-1
12	342343	5	553	40	552.21	-1	40	43200	-1	1	4	1	-1	-1	1	-1	-1
13	342921	102	10027	40	10027	-1	40	43200	-1	1	4	1	-1	-1	1	-1	-1
14	347226	15	33156	8	33157	-1	8	86400	-1	1	5	1	-1	-1	1	-1	-1
15	347593	15	86462	8	86464	-1	8	86400	-1	1	5	1	-1	-1	1	-1	-1
16	347726	5	240379	8	240379	-1	8	345600	-1	-1	5	1	-1	-1	1	-1	-1
17	347844	9	2201	8	2201	-1	8	432000	-1	1	5	1	-1	-1	1	-1	-1
18	347927	48	240135	8	240135	-1	8	432000	-1	-1	5	1	-1	-1	1	-1	-1
19	347985	51	240074	8	240074	-1	8	432000	-1	-1	5	1	-1	-1	1	-1	-1
20	348019	17	4097	8	4097	-1	8	432000	-1	1	5	1	-1	-1	1	-1	-1
21	348036	0	240074	8	240074	-1	8	432000	-1	-1	5	1	-1	-1	1	-1	-1
22	348169	174	12415	8	12415	-1	8	64800	-1	1	5	1	-1	-1	1	-1	-1
25	350187	50	9787	8	9787	-1	8	432000	-1	1	5	1	-1	-1	1	-1	-1
36	353009	165	183	40	182.97	-1	40	43200	-1	1	4	1	-1	-1	1	-1	-1
37	353232	1958	8258	8	8258	-1	8	432000	-1	1	5	1	-1	-1	1	-1	-1
38	353299	119	978	40	978.00	-1	40	43200	-1	1	4	1	-1	-1	1	-1	-1
39	354354	164	550	40	550.04	-1	40	43200	-1	1	4	1	-1	-1	1	-1	-1
58	362040	76953	149115	8	149115	-1	8	432000	-1	-1	5	1	-1	-1	1	-1	-1
59	362323	25	5072	8	5072	-1	8	64800	-1	1	5	1	-1	-1	1	-1	-1
60	364490	16029	2997	40	2997	-1	40	43200	-1	1	4	1	-1	-1	1	-1	-1
61	365580	18058	43215	32	43216	-1	32	43200	-1	1	4	1	-1	-1	1	-1	-1
62	379902	59152	24102	8	24102	-1	8	64800	-1	1	5	1	-1	-1	1	-1	-1
63	379946	59169	148993	8	148993	-1	8	432000	-1	-1	5	1	-1	-1	1	-1	-1
64	384415	54517	31030	30	31030	-1	30	37980	-1	1	1	1	-1	-1	1	-1	-1
65	384420	54512	31152	30	31152	-1	30	37980	-1	1	1	1	-1	-1	1	-1	-1
66	384424	54569	31030	30	31030	-1	30	37980	-1	1	1	1	-1	-1	1	-1	-1
67	384445	85639	31005	30	31005	-1	30	37980	-1	1	1	1	-1	-1	1	-1	-1
68	384483	1050	16245	20	16245	-1	20	22140	-1	1	1	1	-1	-1	1	-1	-1
69	384655	13849	14468	20	14468	-1	20	22140	-1	1	1	1	-1	-1	1	-1	-1
70	384816	17084	12910	20	12910	-1	20	22140	-1	1	1	1	-1	-1	1	-1	-1
71	385030	54024	9355	20	9355	-1	20	22140	-1	1	1	1	-1	-1	1	-1	-1
72	385038	63432	2567	20	2567	-1	20	22140	-1	1	1	1	-1	-1	1	-1	-1

Figure 5.2 HPC2N.swf workload file



58	1	0	4	14400	100	-1	-1	100	14400	-1	1	1	1	-1	1	1	-1	-1
59	2	37	0	14400	100	-1	-1	100	14400	-1	1	1	1	-1	1	1	-1	-1
60	3	76068	9	45	20	-1	-1	20	3600	-1	1	2	2	-1	1	1	-1	-1
61	4	108213	16386	154	1	-1	-1	1	3600	-1	1	3	3	-1	1	1	-1	-1
62	5	108300	16977	50	1	-1	-1	1	3600	-1	1	3	3	-1	1	1	-1	-1
63	6	108336	16596	1112	1	-1	-1	1	3600	-1	1	3	3	-1	1	1	-1	-1
64	7	113136	11796	176777	1000	-1	-1	1000	259200	-1	1	4	4	-1	1	1	-1	-1
65	8	125387	206	45	2560	-1	-1	2560	172800	-1	1	3	3	-1	1	1	-1	-1
66	9	125537	112	353	10000	-1	-1	10000	172800	-1	1	3	3	-1	1	1	-1	-1
67	10	126051	197	42	2560	-1	-1	2560	172800	-1	1	3	3	-1	1	1	-1	-1
68	11	126631	11	2114	1000	-1	-1	1000	259200	-1	1	4	4	-1	1	1	-1	-1
69	12	127033	3	172800	2560	-1	-1	2560	172800	-1	1	3	3	-1	1	1	-1	-1
70	13	127085	4	49	1	-1	-1	1	3600	-1	1	3	3	-1	1	1	-1	-1
71	14	127203	8	42	1	-1	-1	1	3600	-1	1	3	3	-1	1	1	-1	-1
72	15	128429	4	53	1	-1	-1	1	3600	-1	1	3	3	-1	1	1	-1	-1
73	16	128531	1	42	1	-1	-1	1	3600	-1	1	3	3	-1	1	1	-1	-1
74	17	128608	8	657	1	-1	-1	1	3600	-1	1	3	3	-1	1	1	-1	-1
75	18	128780	0	180	1000	-1	-1	1000	259200	-1	1	4	4	-1	1	1	-1	-1
76	19	128990	0	365	1000	-1	-1	1000	259200	-1	1	4	4	-1	1	1	-1	-1
77	20	129533	0	354	1000	-1	-1	1000	259200	-1	1	4	4	-1	1	1	-1	-1
78	21	129582	4	374	1	-1	-1	1	3600	-1	1	3	3	-1	1	1	-1	-1
79	22	129980	243	31	1000	-1	-1	1000	259200	-1	1	4	4	-1	1	1	-1	-1
80	23	130135	11	327	1000	-1	-1	1000	259200	-1	1	4	4	-1	1	1	-1	-1

Figure 5.3 KIT-FH2-2016-1.swf workload file

49	11	566129	5	28826	1	27758	-1	1	28800	-1	5	153	75	18180	3	-1 -1 -1
50	12	566290	532	26171	1	24666	-1	1	28800	-1	1	153	75	18184	3	-1 -1 -1
51	13	567314	15757	8071	8	7334	-1	8	64800	-1	1	150	6	13592	4	-1 -1 -1
52	14	571164	21568	64832	32	44134	-1	32	64800	-1	5	6	6	13602	3	-1 -1 -1
53	15	574294	4647	64384	7	58033	-1	7	64800	-1	1	151	75	13607	3	-1 -1 -1
54	16	576100	6	19000	1	18830	-1	1	43200	-1	1	152	75	13610	3	-1 -1 -1
55	17	577685	3793	118561	5	117278	-1	5	172800	-1	1	199	86	18261	3	-1 -1 -1
56	18	578846	8	24861	1	23875	-1	1	28800	-1	1	153	75	13613	3	-1 -1 -1
57	19	580157	20861	-1	-1	-1	-1	38	6900	-1	5	189	75	18288	5	-1 -1 -1
58	20	581034	2038	10979	1	10894	-1	1	18000	-1	1	154	75	13634	4	-1 -1 -1
59	21	581434	2139	16674	11	16258	-1	11	36000	-1	1	160	75	18301	3	-1 -1 -1
60	22	582841	1199	8385	32	1536	-1	32	14400	-1	1	150	6	13641	2	-1 -1 -1
61	23	583532	21288	16793	11	16264	-1	11	36000	-1	1	160	75	18305	3	-1 -1 -1
62	24	584165	16591	3634	24	3504	-1	24	3600	-1	5	133	50	13646	3	-1 -1 -1
63	25	584785	4129	1229	2	1089	-1	2	1200	-1	5	34	7	13651	3	-1 -1 -1
64	26	584801	10510	2207	16	1195	-1	16	3600	-1	1	7	7	234	3	-1 -1 -1
65	27	584826	6324	40	8	7.38	-1	8	1200	-1	1	34	7	13662	3	-1 -1 -1
66	28	584832	6361	311	8	259.00	-1	8	2400	-1	1	34	7	13665	3	-1 -1 -1
67	29	584838	6677	1228	8	1157	-1	8	1200	-1	5	34	7	13668	3	-1 -1 -1
68	30	584875	10119	47	16	6.00	-1	16	1200	-1	1	34	7	13671	3	-1 -1 -1
69	31	584919	10126	213	16	156.94	-1	16	1200	-1	1	34	7	13674	3	-1 -1 -1

Figure 5.4 SDSC-SP2-1998-4.2-cln.swf workload file

The value of each column's description is showed in table 5.1.

Additionally, Alea [15] [16] [17] utilizes its own machine-description format for the machines files (resources), which are stored in files with the (.machines) filename extension. One computer cluster is specified per line in the file format, with TAB space used to separate attributes. The characteristics of one line are as follows:

```
cluster_id   cluster_name  number_of_nodes   CPUs_per_node
CPU_speed   RAM_per_node_in_KB
```

(cluster_id,cluster_name,number_of_nodes,CPUs_per_node,
CPU_speed,RAM_per_node_in_KB)

```
ant -f C:\\Users\\User\\Desktop\\Research\\alea(master)\\Alea -Dnb.internal.action.name=run run
init:
Deleting: C:\Users\User\Desktop\Research\alea(master)\Alea\build\built-jar.properties
deps-jar:
Updating property file: C:\Users\User\Desktop\Research\alea(master)\Alea\build\built-jar.properties
compile:
run:
Working directory: C:\Users\User\Desktop\Research\alea(master)\Alea
result root: results\2024-06-25-14-54-52
Initialising SimJava2 modified by Dalibor Klusacek (xklusac@fi.muni.cz).
==================End=============
ExperimentSetup.local_schedulers.size()0
Now scheduling 3100 jobs by: Improved min-min and priority, using hpc2n.swf data set.
Starting Machine Loader ...
Opening: ./data-set/LYK/hpc2n.swf.machines
The system has 240 CPUs and 120 GBs of RAM.
hpc2n.swf_JobLoader: openning all jobs from hpc2n.swf
Opening job file at: ./data-set/LYK/hpc2n.swf
Opening job file at SWF Loader:total_jobs LYK ::3100:240:1:1:hpc2n.swf
Starting simulation using Alea 4.0
Starting GridSim version 5.0
Entities started.
GridResource/Cluster count: 1
Resource Info ExperimentSetup.local_schedulers.size():1
0th allocation policy 'AdvancedSpaceSharedPolicy' is used for the resource no. 9
============================================================================================
List of resources:
id = 9, name = seth, CPUs = 240, CPU rating = 1, machines = 120, props=0, RAM=1024.0 MB
Total available MIPS power = 240.0 MIPS in 240.0 CPUs and machines = 120
============================================================================================
```

Figure 5.5 Simulation result

5.2 Graphical User Interface of the Proposed System

First, the user selects the suggested algorithm in the simulator settings. Figure 5.4 displays the outcome metrics and the data set (workload log file). Alea 2 is an event-based modular simulator akin to GridSim, consisting of independent parts that cooperate to generate the necessary simulation functionalities. The centralized scheduler, grid resource(s) with the local task allocation policy, job loader, machine and failure loader, and additional classes in charge of configuring the simulation, viewing it, and generating simulation output make up this system. Grid users are not simulated directly at this time, however the task loader entity can be used as a parent class for a future Grid user implementationThe simulator's behavior is managed using the event-based message forwarding protocol. Every simulated event—such as the arrival of the task or its completion—is defined by a single message. It contains the recipient's identify, the kind of event, the time of the event, and the message content. A message such as this might appear in the event of, say, a job arrival (scheduler ID; job arrival event; job arrival time; job description). The simulator is fully compatible with the latest GridSim 5.0 release, since no changes have been made to the GridSim program itself. Every extension was created by extending the default GridSim (parent) classes through the implementation of child classes. Similarly, the object-oriented approach used by Alea 2 and GridSim makes it simple to extend existing features.

The proposed system is using Alea 4 simulator. Alea 4 is additional version of Alea 2 [16].

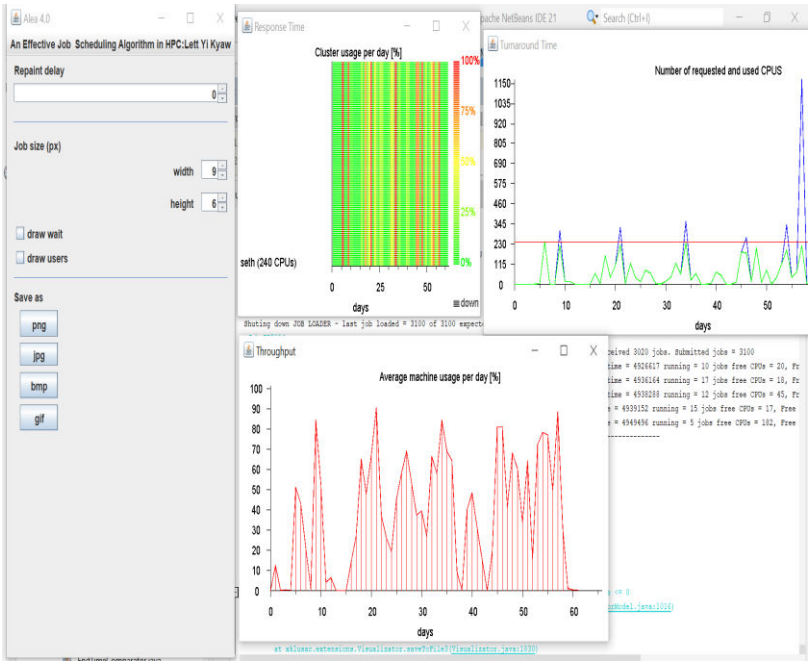


Figure 5.6 The Simulator

When the user chooses algorithm from the system and clicks the run button, the system scheduled the job of input log file (workload file) from the dataset. Then, it displays the performance metrics of input scheduling algorithm. After the simulation is complete, the results are used as an input to create the graphs. A new set of scheduling has started. When all jobs that have been submitted have been completed and no new jobs arise, the cycle is over. The simulation then comes to a close, and the output files contain the results.

5.3 Summary

This chapter is the main point of the whole research. This chapter explained the input of the system, their processing states and produces the results of the system. The simulation of the Alea simulator is input user desire algorithm, input log files and evaluate the QoS metrics (throughput, turnaround time and response time). When the user selects an algorithm and clicks the run button, the system schedules the job from the input log file (workload file) within the dataset. After scheduling, it displays

the performance metrics of the chosen algorithm. Once the simulation finishes, graphs are generated using the simulation results. A new scheduling round begins, and this cycle continues until no new jobs arrive and all submitted jobs are completed. Finally, the simulation ends, and the results are stored in the output files.

CHAPTER 6
EXPERIMENTAL RESULTS

High-Performance Computing (HPC) systems require effective job scheduling algorithms to manage the allocation of resources and maximize system utilization. This research proposes a novel job scheduling algorithm designed specifically for HPC environments. The algorithm, named an effective job scheduling algorithm, aims to optimize system throughput, minimize job completion times, and ensure fair resource allocation among competing tasks.

The proposed system showed the performance evaluation of the proposed algorithm using a variety of benchmark workloads on a state-of-the-art HPC cluster. The experimental results demonstrate that the proposed algorithm outperforms existing scheduling algorithms in terms of throughput, job completion times, and fairness.

Overall, the proposed algorithm is a promising solution for job scheduling in HPC environments, offering significant improvements in system performance and resource utilization. The proposed algorithm lays the groundwork for future research in the field of HPC scheduling, with potential applications in scientific computing, big data analytics, and other compute-intensive domains.

6.1 Experimental Results

The experimental work for the proposed algorithm is done using Alea [15][16], a simulator, based on the latest version of the GridSim toolkit, GridSim 5, to simulate and model HPC and application environment. Alea is using Java language and contains scheduling algorithms, various objective functions and the support for configuring machine characteristics.

In the first experiment, the complex data from High Performance Computing Center North, Sweden will be used to demonstrate the simulation capability of the simulator[10]. These data allow us to perform very realistic simulations, involving all characteristics of workload format. The experiment involves 527371 jobs that were originally executed StartTime: Sun Jul 28 09:04:05 CEST 2002 to EndTime: Mon Jan 16 20:31:24 CET 2006. Through the experiment, different scheduling algorithms will be compared: FCFS, EASY Backfilling. The second experiment, we will use the

complex data from HPC Systems group of the San Diego Supercomputer Center (SDSC) will be used which is the leading-edge site of the National Partnership for Advanced Computational Infrastructure (NPACI) [92].

The third experiment, input workload from Mehmet Soysal, Karlsruhe Institue of Technology / Steinbuch Centre for Computing will be inserted to the simulator [15]. In this workload included 114355 jobs that were executed time zone (Europe/Berlin) StartTime: Wed Jun 01 02:12:45 CEST 2016 to EndTime: Thu Jan 04 14:31:24 CET 2018.Three of these experiments will show the performance of HPC system are evaluated by Quality of Service(QoS) metrics (turnaround time, throughput and response time) with the proposed system. The Alea 4 has been extended for use in our work on cluster and grid scheduling, where it has been able to simulate scheduling under a variety of setups using a range of data sets and objective functions. By making small changes to the current simulator, the basic functionality of the scheduling methods or optimization criteria can be added to the modular simulator.

The performance of the proposed system is measured with three HPC quality measurement methods – response time, turnaround time and throughput.

- **Throughput** - The number of processes that are completed by a system per time unit. Response time is the time spent when the process is in the ready state and gets the CPU for the first time.
- **Turnaround Time** - Waiting time plus execution time. In other words, it is the delay between job submission and job completion.
  - CT (Completion Time) – Completion time is the exact time when a process finishes the execution.
  - AT (Arrival Time) – Arrival time is called as the time of arrival of a process before the state of preparedness (before its execution).
  - Therefore, CT – AT = TAT. - ---------- Eq. (6.1)
  -
- **Response Time** –The duration between the submission of the job and its completion, or equivalently its waiting time and its length.
  - RT=T(wait)+T(real) ---------- Eq.(6.2)

    Results are showed in figure 6.1 to 6.10.

6.1.1 FCFS

HPC2n.swf file is inserted into simulator and the algorithm is choosing FCFS, the virtualization graph is showed as in Figure 6.1. The throughput result is showed.

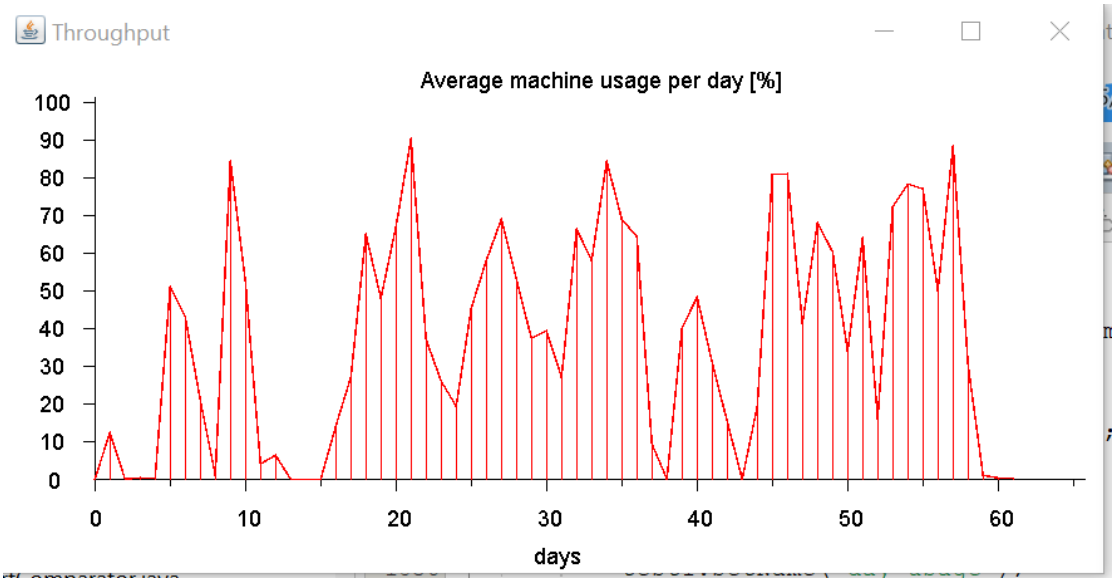


Figure 6.1 Throughput [FCFS, using hpc2n.swf data set]

Figure 6.2 showed the turnaround time of the whole log file using this simulator.

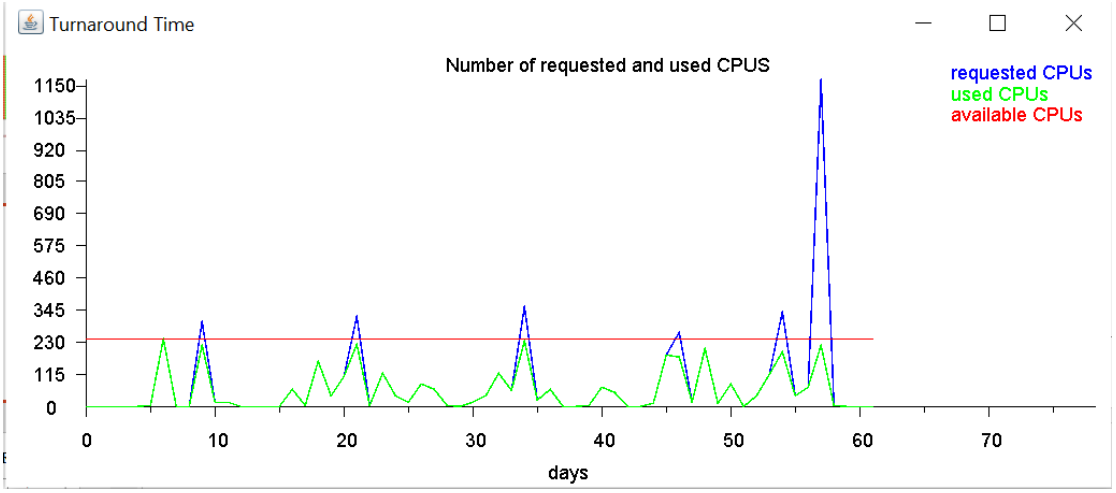


Figure 6.2 Turnaround Time [FCFS, using hpc2n.swf data set]

Response is very important for the performance of the system. So in this research are calculated the response time of the system depend on user desired algorithm which is showed in figure 6.3.

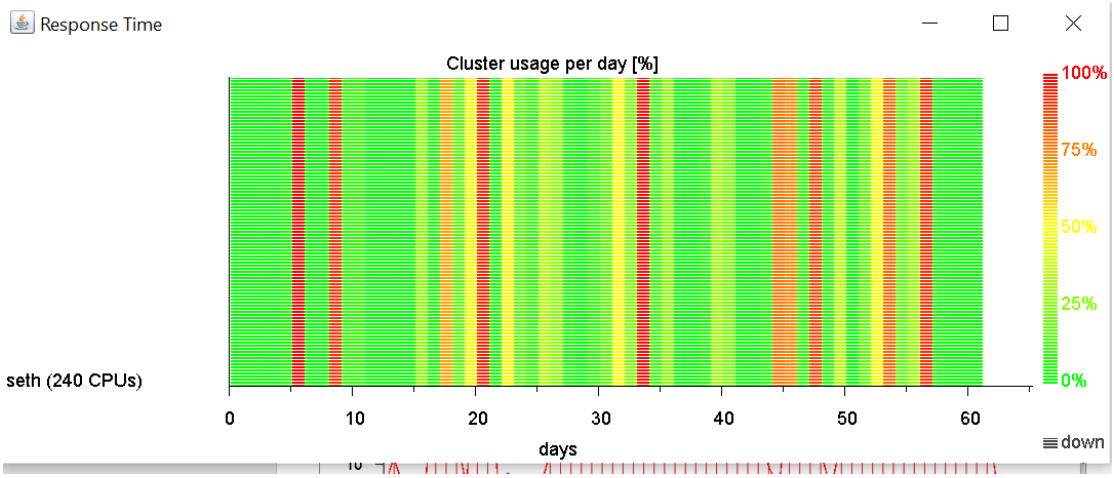


Figure 6.3 Response Time [FCFS, using hpc2n.swf data set]

Figure 6.4 showed three metrics calculation using the proposed algorithm with the above data set.

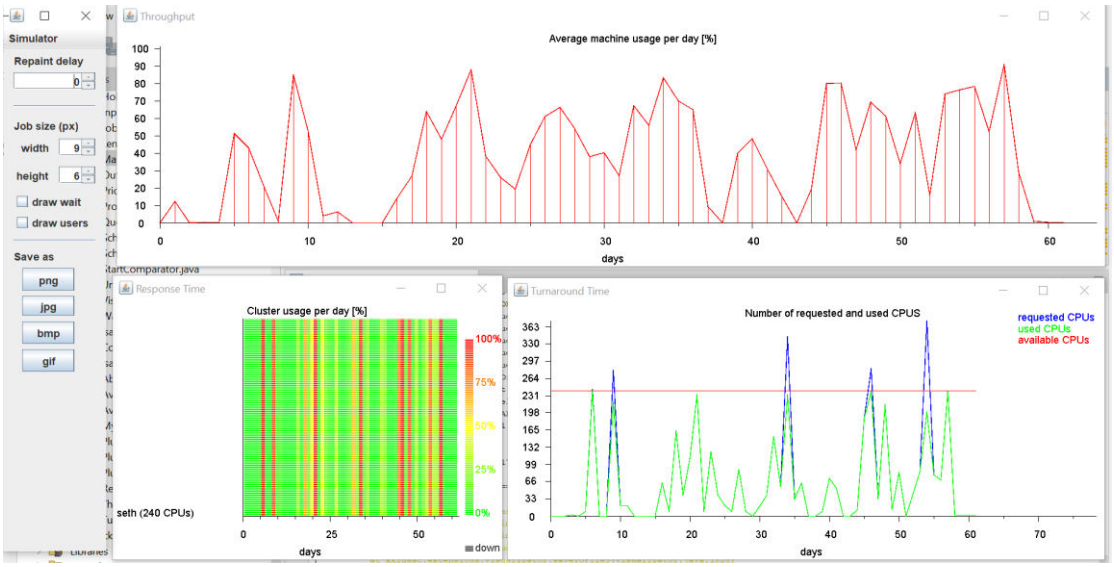


Figure 6.4 Throughput, Turnaround Time ,Response Time [proposed algorithm, using hpc2n.swf data set]

Figure 6.5, 6.6 and 6.7 are the results of the proposed algorithm.

6.1.2 Proposed System

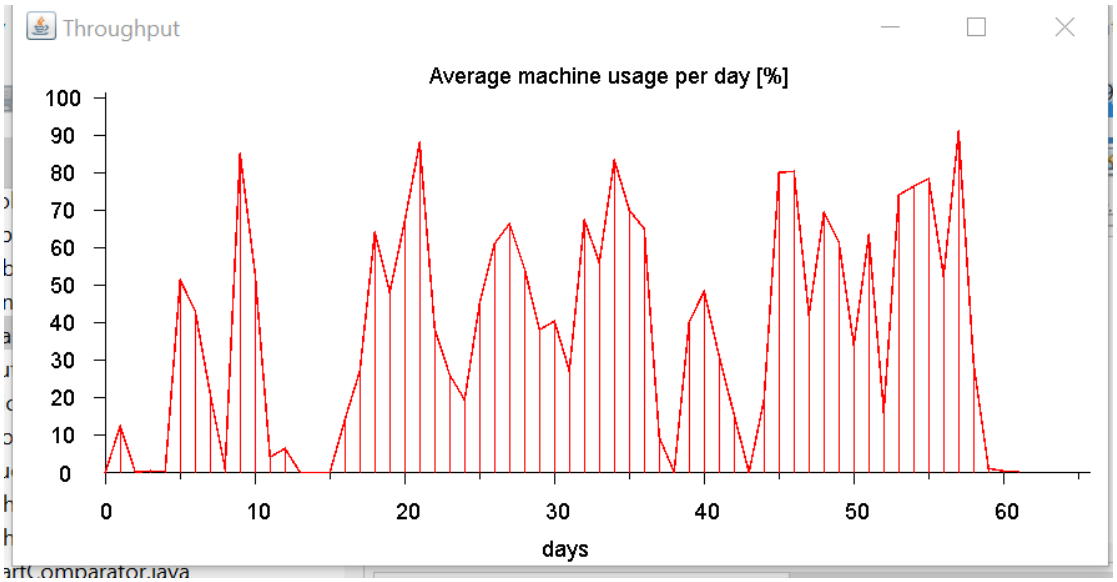


Figure 6.5 Throughput [proposed algorithm, using hpc2n.swf data set]

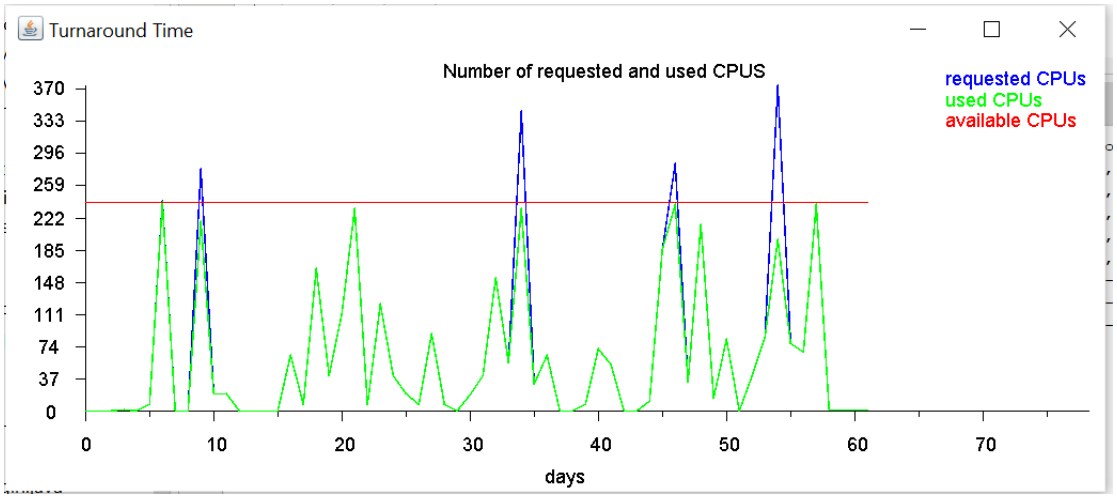


Figure 6.6 Turnaround Time [proposed algorithm, using hpc2n.swf data set]

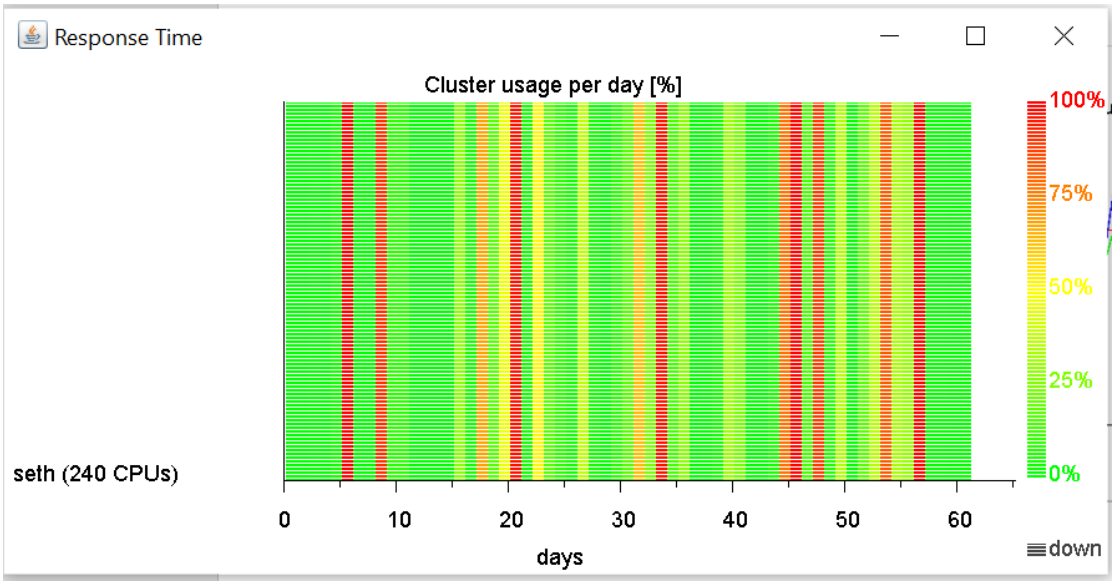


Figure 6.7 Response Time [proposed algorithm, using hpc2n.swf data set]

6.1.3 EASY Backfilling

EASY Backfilling is useful for compare with the proposed system in HPC environment. User can get the throughput results as in figure 6.8, turnaround time results similar figure 6.9 and response time as in figure 6.10.

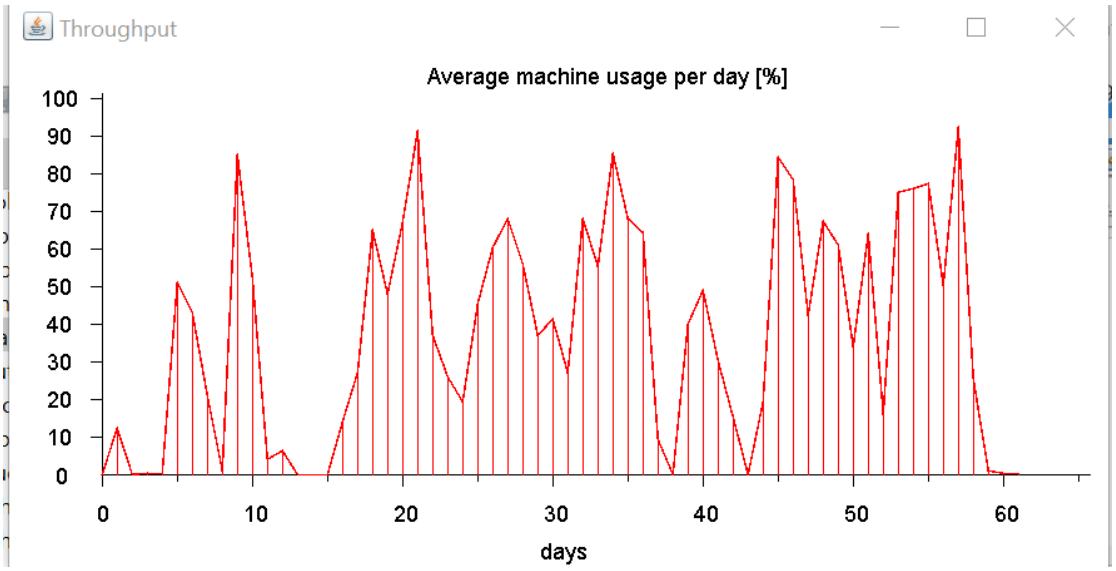


Figure 6.8 Throughput [EASY Backfilling, using hpc2n.swf data set]

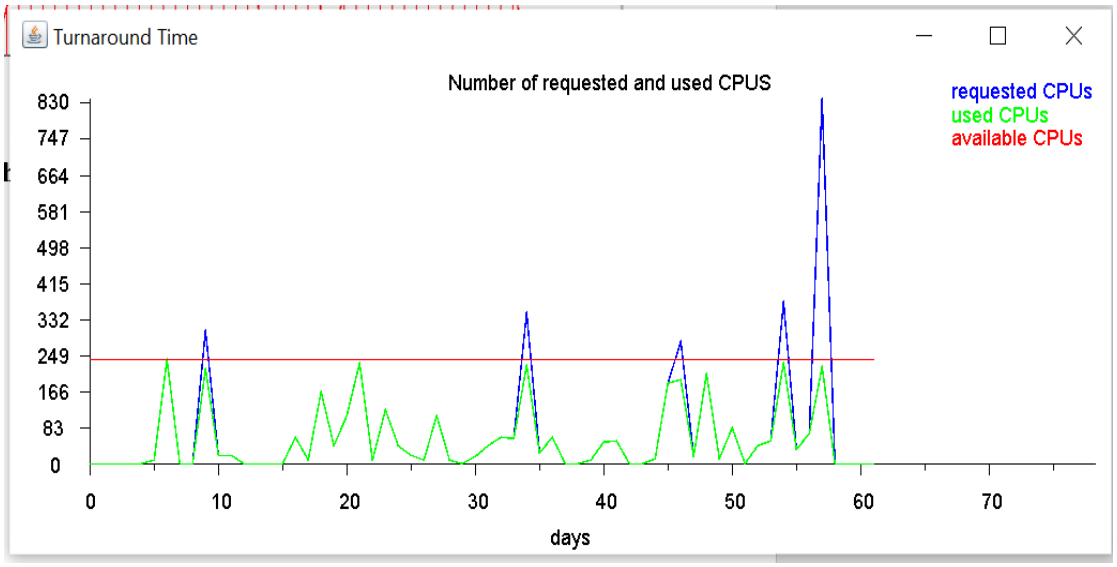


Figure 6.9 Turnaround Time [EASY Backfilling, using hpc2n.swf data set]

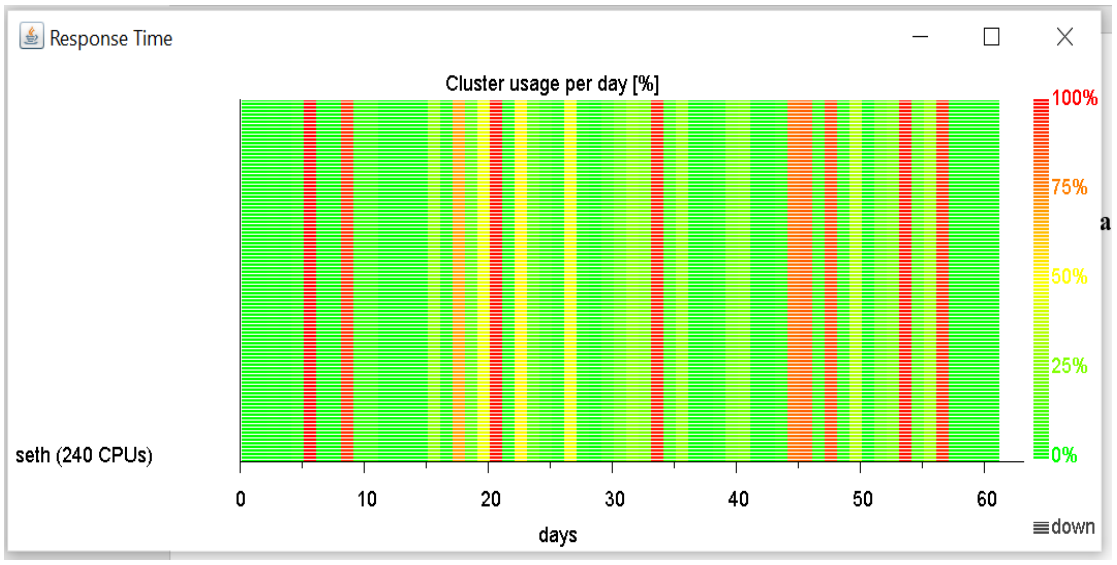


Figure 6.10 Response Time [EASY Backfilling, using hpc2n.swf data set]

6.2 Comparison Results

The following table showed the results of the FCS, EASY Backfilling and the proposed system.

Data Set	Throughput			Turnaround Time			Response Time		
	FCFS	EASY Backfilling	Proposed Algorithm	FCFS	EASY Backfilling	Proposed Algorithm	FCFS	EASY Backfilling	Proposed Algorithm
HPC2N	13	28	24	30	33	7	30	30	8
KIT-FH2-2016-1	22	7	14	29	30	14	25	29	12
SDSC-SP2-1998-4.2-cln	23	14	25	13	14	15	19	21	17

Table 6.1 Comparison results of FCFS, EASY Backfilling , Proposed Algorithm

6.3 Summary

This chapter showed the proposals of a new job scheduling algorithm tailored for HPC environments, aiming to optimize system throughput, minimize job completion times, and ensure fair resource allocation. The proposed algorithm's performance was evaluated using various benchmark workloads on a state-of-the-art HPC cluster, demonstrating superior results compared to existing algorithms. Throughput, turnaround time, and response time were the key metrics used to evaluate the quality of the proposed method. The Alea simulator, which is built on the GridSim toolkit, was used in the experimental work to represent the HPC environment and application scenarios. The proposed algorithm offers significant improvements in system performance and resource utilization, laying the groundwork for future research in HPC scheduling and potential applications in scientific computing, big data analytics, and other compute-intensive domains.

CHAPTER 7
CONCLUSION AND FUTURE WORK

The contribution of the research work is that Scheduling Jobs can be divided into different categories according to user needs, and then the best running time can be set for each task based on different goals. The task of scheduling QoS in a HPC environment will be vastly improved indirectly. By incorporating more metrics, a better scheduling algorithm can be constructed from current techniques, resulting in outstanding results and outputs that are likely to be deployed in the future in a cloud situation with HPC. Many researchers are searching, but in many applications in HPC and Cloud, the research can be carried out in many applications until makespan remains in issues.

7.1 Summary

This study has focused on the creation of dynamic job scheduling methods for high-performance computing. While static scheduling for HPC has seen significant advances, dynamic scheduling has not experienced the same level of development since its inception. This is primarily because, in the past, dynamic scheduling has never been a necessary requirement. However, given the recent advances in HPC, the impetus for further research and the development of workable solutions stems primarily from three sources: (i) scientific applications that explore new domains with potentially variable resource requirements across application phases; (ii) the increasing ease with which malleability can be realized in upcoming cluster systems; and (iii) the investigation of heterogeneous architectures with various network-attached resources, such as storage and accelerators. Thus, the primary goal of this study has been to present new and useful approaches for resource management and dynamic scheduling that may be included into batch production systems.

This chapter presents a novel job scheduling approach for HPC environments, where jobs are categorized based on user needs and optimal running times are set to improve task scheduling QoS. The research envisions the application of the proposed system in real-world HPC environments, emphasizing its potential impact on scientific research, engineering, and industrial applications. Despite its transformative impact, HPC faces challenges like power consumption, system resilience, and data

management. Future research aims to address these issues and explore emerging technologies like quantum computing and scaling up computing, promising further advancements and broader accessibility through cloud-based services and open-source frameworks

7.2. Discussion

This research paper has discussed empirical studies of the proposed method for HPC. In summary, the significant empirical results that highlight the quality factors of HPC system. The design of the proposed system is better than the existing methods. It reduces the processing time than the other method while analyzing the data which are changing over time and it is not useful in reality.

7.3 Advantages and Limitations of the Proposed System

The proposed system provides performance using simulator. This system improved the performance than existing algorithm (FCFS, EASY Backfilling). Since practically all computer resources are scheduled before usage, scheduling is an essential operating system function. One of the main components of a computer is the CPU. By dividing up the CPU among multiple tasks (jobs), central processing unit (CPU) scheduling is crucial. The operating system has the power to increase a computer's productivity, but the CPU is still the most crucial component. The scheduling goal is to maximize CPU utilization by allowing the maximum number of processes to execute continuously and to get fastest response time. The creation of excellent scheduling algorithms that meet scheduling objectives is the foundation of a highly efficient scheduler. Easy, First-Come, First-Served The two fundamental scheduling strategies for a high-performance computing (HPC) system, backfilling and optimization, were analyzed, evaluated, and contrasted to demonstrate which algorithm is better suited for specific processes and how their scheduling is carried out. The proposed system produces effective job scheduling technique.

7.4 Future Work

The proposed system will be used High-Performance Computing (HPC) systems stand as monumental pillars of computational capability, driving innovation across scientific research, engineering, and industrial domains at real world not in

simulator. Throughout this exploration, we have delved into the architecture, capabilities, applications, and challenges of HPC systems, shedding light on their transformative impact on modern computing.

From the intricate interplay of processing cores and high-speed networks to the utilization of specialized accelerators such as GPUs and FPGAs, the architecture of HPC systems embodies a harmonious convergence of hardware and software engineering. This synergy enables HPC systems to tackle computation-intensive tasks with unparalleled speed and efficiency, unlocking new frontiers in scientific discovery and technological advancement.

The applications of HPC systems are as diverse as they are profound, spanning disciplines from computational fluid dynamics and molecular modeling to climate simulation and financial forecasting. These systems empower researchers and practitioners to simulate complex phenomena, analyze vast datasets, and solve optimization problems at scales previously unimaginable, paving the way for groundbreaking discoveries and innovations.

However, the journey of HPC is not without its challenges. As when the boundaries of computational capability are pushed, there are confrontations with issues of addressing these challenges requires ongoing research and innovation in areas such as energy-efficient computing, fault tolerance, and scalable storage solutions.

Looking ahead, the future of HPC holds promise and excitement. Emerging technologies such as quantum computing, neuromorphic computing, and exascale computing are poised to usher in a new era of computational capability, enabling even greater advances in science, engineering, and society at large. Moreover, the democratization of HPC through cloud-based services and open-source software frameworks is making high-performance computing more accessible to researchers and organizations of all sizes, further fueling innovation and collaboration. The approach promises to enhance current techniques by incorporating more metrics, ultimately benefiting cloud-based HPC applications in the future. The thesis focuses on developing dynamic job scheduling techniques for HPC, a field that has lagged behind static scheduling advancements. The motivation for dynamic scheduling comes from the evolving demands of scientific applications, the malleability of new cluster systems, and the exploration of heterogeneous architectures. Research findings indicate that the suggested dynamic scheduling approach surpasses current

techniques, resulting in shorter processing times for time-sensitive data examination. This method provides significant quality improvements in HPC systems. The proposed system has limitations such jobs are huge amount; the processing of the simulator will delay. The simulator configuration file will be set up by the user according to input. The (.machines) files will created before using the proposed system and the simulator.

Author's Publications

1. Lett Yi Kyaw, Sabai Phyu, "Job Scheduling for High Performance Computing and Cloud Environment", The 17th International Conference on Computer Applications 2019 (ICCA 2019), 2019. Pg 48-52.
2. Lett Yi Kyaw, Sabai Phyu, "Scheduling Methods in HPC System: Review", The IEEE 18th International Conference on Computer Applications 2020 (ICCA 2020) ,2020. Pg 78-86 .
3. Lett Yi Kyaw, Kyar Nyo Aye, "Modelling and Simulation of Min-Min and Priority Algorithm in HPC Environment", IJCSE, Volume 14 Issue 3 May-June 2024, e-ISSN:0976-5166,p-ISSN:2231-3850,Vol.15,No.3,2024. DOI: 10.21817/indjcse/2024/v15i3/241503048.

Bibliography

[1] A. B Downey, “A parallel workload model and its implications for processor allocation”, Cluster Computing, 1(1):133–145, 1998.

[2] A. Fernández, V. Beltran, X. Martorell, R. M. Badia, E. Ayguadé, J. Labarta,” Task-Based Programming with OmpSs and Its Application”, In Euro-Par 2014: Parallel Processing Workshops, volume 8806 of Lecture Notes in Computer Science, pages 601–612,Springer International Publishing, 2014.

[3] A. Fernández, V. Beltran, X. Martorell, R. M. Badia, E. Ayguadé, J. Labarta,” Task-Based Programming with OmpSs and Its Application”, In Euro-Par 2014: Parallel Processing Workshops, volume 8806 of Lecture Notes in Computer Science, pages 601–612, Springer International Publishing, 2014.

[4] A. Jokanovic, J. C. Sancho, G. Rodriguez, A. Lucero, C. Minkenberg, and J. Labarta,” Quiet neighborhoods: Key to protect job performance predictability”, In International Parallel and Distributed Processing Symposium, May 2015.

[5] A. Maurya, B. Nicolae, I. Guliani, M. M. Rafique,”CoSim: A Simulator for Co-Scheduling of Batch and On-Demand Jobs in HPC Datacenters”, University of Glasgow, November 01,2020.

[6] A. Varga, “The OMNeT++ Discrete Event Simulation System”, Proceedings of the European Simulation Multiconference (ESM 2001), Prague, Czech Republic, June 6-9, 2001.

[7] B. Barney, “Slurm and moab” https://computing.llnl.gov/tutorials/ moab, August 2017. Retrieved August 22, 2017.

[8] B. Gan, S. Huang,” Scheduling dynamically evolving parallel programs using the genetic approach”, In The Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region, May 2000.

[9] Bergman, K., Borkar, S., Campbell, D., Carlson, W., Dally, W., Denneau, M., Franzon, P., Harrod, W., Hill, K., Hiller, J., et al.,” Exascale computing study: Technology challenges in achieving exascale systems”, Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep 15 (2008), 2008.

[10] C. Galleguillos, Z. Kiziltan, A. Netti and R. Soto, “AccaSim: a customizable

workload management simulator for job dispatching research in HPC systems", Springer, 01 February 2019.

[11] C. Huang, O. Lawlor, L. V. Kalé," Adaptive MPI", In Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2003), LNCS 2958, October 2003.

[12] C. Klein, C. Perez, "An RMS for Non-predictably Evolving Applications", In 2011 IEEE International Conference on Cluster Computing (CLUSTER), September 2011.

[13] CACI, "Simscript: a simulation language for building large-scale, complex simulation models", CACI Products Company, San Diego, CA, USA, http://www.simscript.com/simscript.cfm

[14] D. Abramson, J. Giddy, L. Kotler, "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?", Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2000), May 1-5, IEEE Computer Society (CS) Press ,Cancun, Mexico , USA ,2000.

[15] D. Klusáček, G. Podolníková, Š. Tóth," Complex Job Scheduling Simulations with Alea 4", In Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques, pages 124-129, 2016.

[16] D. Klusáček, H. Rudová,"Alea 2 - Job Scheduling Simulator", In proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010), ICST, 2010.

[17] D. Klusáček, M.t Soysal ,F. Suter,"Alea - Complex Job Scheduling Simulator", In Proceedings of PPAM 2019: Parallel Processing and Applied Mathematics, pages 217–229, 2019.

[18] D. Kumar, Z. Shae, H. Jamjoom," Scheduling Batch and Heterogeneous Jobs with Runtime Elasticity in a Parallel Processing Environment", In Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International, May 2012.

[19] D., J.," Report on the Sunway Taihulight system", www. netlib. org. Retrieved June 20 (2016), 2016.

[20] E. Walker,"Benchmarking Amazon EC2 for High-Performance Scientific Computing", October 2008.http://www.usenix.org/publications/login/2008-10/benchmark_results.tgz.

[21] Elmroth, E., Gardfj¨all, P.,"Design and evaluation of a decentralized system for Grid-wide fairshare scheduling", In H. Stockinger et al, ed.: Proceedings of e-Science 2005, IEEE CS Press (2005) 221–229, 2005.

[22] F. Howell, R. McNab," SimJava: A Discrete Event Simulation Package For Java With Applications In Computer Systems Modelling" , First International Conference on Web-based Modelling and Simulation, San Diego, CA, Society for Computer Simulation, January 1998.

[23] Foster, I., Kesselman, C., "The grid: blueprint for a new computing infrastructure", Morgan Kaufmann (2004), 2004.

[24] Foster, Y. Zhao, I. Raicu, S. Lu,"Cloud computing and grid computing 360-degree compared", In Proceedings of Grid Computing Environments Workshop, pages 1–10, 2008.

[25] G. Mounie, C. Rapine, D. Trystram, "Efficient Approximation Algorithms for Scheduling Malleable Tasks", Proc. of the Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures. ACM, 1999.

[26] G. Sabin, M. Lang, P. Sadayappan, "Moldable Parallel Job Scheduling Using Job Efficiency: An Iterative Approach", Eitan Frachtenberg and Uwe Schwiegelshohn, editors, Job Scheduling Strategies for Parallel Processing, volume 4376 of Lecture Notes in Computer Science, pages 94–114. Springer Berlin Heidelberg, 2007.

[27] G. Utrera, S. Tabik, J. Corbalan, J. Labarta, " A Job Scheduling Approach for Multi-core Clusters Based on Virtual Malleability", Euro-Par 2012 Parallel Processing, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012.

[28] H. Casanova, A. Giersch, A. Legrand, M. Quinson, F. Suter," Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms", Journal of Parallel and Distributed Computing, 74(10):2899–2917, June 2014.

[29] H. Casanova," Simgrid: A Toolkit for the Simulation of Application Scheduling", Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001), May 15-18, Brisbane, Australia, IEEE Computer Society Press, USA, 2001.

[30] H. Lederer, G. J. Pringle,"DEISA: Extreme Computing in an Advanced

Supercomputing Environment", Conference: Parallel Computing: Architectures, Algorithms and Applications, ParCo 2007, Forschungszentrum Jülich and RWTH Aachen University, Germany, 4-7 September 2007.

[31] H. Sun, Y. Cao, W. Hsu,"Fair and Efficient Online Adaptive Scheduling for Multiple Sets of Parallel Applications", In IEEE 17th International Conference on Parallel and Distributed Systems, 2011.

[32] H. Topcuoglu, S. Hariri, M. Wu,"Performance-eective and low-complexity task scheduling for heterogeneous computing", IEEE transactions on parallel and distributed systems 13, 3 (2002), 260–274, 2002.

[33] https://accasim.readthedocs.io/en/latest/.

[34] https://batsim.readthedocs.io/en/latest/.

[35] https://link.springer.com/article/10.1007/s10586-023-04060-4.

[36] https://www.metacentrum.cz/en/devel/simulator/

[37] https://www.researchgate.net/figure/Evolution-of-HPC-Platforms-for-Performance_fig1_343485723.

[38] I. Foster, C. Kesselman (editors), The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann Publishers, USA, 1999.

[39] J. Blazewicz, M. Machowiak, G. Mounié, D. Trystram, "Approximation Algorithms for Scheduling Independent Malleable Tasks", Euro-Par 2001 Parallel Processing, Lecture Notes in Computer Science,Springer Berlin Heidelberg, 2001.

[40] J. Buisson, F. André, J. Pazat,"A Framework for Dynamic Adaptation of Parallel Components", In Proc. of the International Conference on Parallel Computing (ParCo) 2005.

[41] J. Buisson, O. Sonmez, H. Mohamed, W. Lammers, D. Epema, " Scheduling malleable applications in multicluster systems", IEEE International Conference on Cluster Computing, September 2007.

[42] J. Buisson, O. Sonmez, H. Mohamed, W. Lammers, D. Epema," Scheduling malleable applications in multicluster systems", In 2007 IEEE International Conference on Cluster Computing, September 2007.

[43] J. Hines,"Moab scheduling tweak tightens titan's workload", https://www.olcf.ornl.gov/2015/10/13/moab-scheduling-tweak-tightens-titans-workload/, Oct 2015. Retrieved July 18, 2018.

[44] J. Hungershofer," On the Combined Scheduling of Malleable and Rigid jobs", 16th Symposium on Computer Architecture and High Performance Computing, Oct 2004.

[45] J. Patel, A. K. Solanki, "CPU Scheduling: A Comparative Study", Proceedings of the 5th National Conference, IndiaCom-2011, Computing for Nation development, 2011.

[46] J. Yeom, A. Bhatele, K. R. Bisset, E.Bohm, A. Gupta, L. V. Kalé, M. Marathe, D. S. Nikolopoulos, M. Schulz, L. Wesolowski, "Overcoming the scalability challenges of epidemic simulations on blue waters", In Proceedings of the IEEE International Parallel & Distributed Processing Symposium, IPDPS '14. IEEE Computer Society, May 2014.

[47] J. Yu, R. Buyya, "A taxonomy of scientist workflow systems for grid computing", ACM Sigmod Record 34, 3 (2005), 44–49, 2005.

[48] Jackson, D., Snell, Q., Clement, M.," Core algorithms of the maui scheduler", Job Scheduling Strategies for Parallel Processing, Springer (2001) 87–102, 2001.

[49] K. E. Maghraoui, T. J. Desell, Boleslaw K. Szymanski, and Carlos A. Varela. Malleable Iterative MPI Applications. Concurrency and Computation: Practice and Experience, 21, 2009.

[50] K. Huang, T. Huang, M. Tsai, H. Chang," Moldable Job Scheduling for HPC as a Service. In James J. (Jong Hyuk) Park, Ivan Stojmenovic, Min Choi, and Fatos Xhafa, editors, Future Information Technology, volume 276 of Lecture Notes in Electrical Engineering, pages 43–48, Springer Berlin Heidelberg, 2014.

[51] Kay, J., Lauder, P.," A fair share scheduler", Communications of the ACM 31 (1) (1988) 44–55, 1988.

[52] L. F. G´oes, P. Guerra, B. Coutinho, L. Rocha, W. Meira, R. Ferreira, D. Guedes, W. Cirne," AnthillSched: A Scheduling Strategy for Irregular and Iterative I/O-Intensive Parallel Jobs", In Job Scheduling Strategies for Parallel Processing (JSSPP), volume 3834 of Lecture Notes in Computer Science, pages 108–122. Springer Berlin Heidelberg, 2005.

[53] L.V. Kalé , S. Krishnan, "CHARM++: A Portable Concurrent Object Oriented System Based on C++ ", In Proceedings of OOPSLA'93, ACM

Press, September 1993.

[54] L.V. Kalé, S. Kumar, J. DeSouza, " A Malleable-Job System for Timeshared Parallel Machines", In 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid), May 2002.

[55] M. A. Jette, A.y B. Yoo, M. Grondona, " SLURM: Simple Linux Utility for Resource Management", In In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP),Springer-Verlag, 2003.

[56] M. Baker, R. Buyya,D. Laforenza, " The Grid: International Efforts in Global Computing", International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet, July 31 - August 6, Rome, Italy ,2000.

[57] M. C. Cera, Y. Georgiou, O. Richard, N. Maillard, P. O. A. Navaux, "Supporting Malleability in Parallel Architectures with Dynamic CPUSETs Mapping and Dynamic MPI", In Proceedings of the 11th International Conference on Distributed Computing and Networking. Springer-Verlag, 2010.

[58] M. C. Cera, G. P. Pezzi, E. N. Mathias, N. Maillard, P. O. A. Navaux, " Improving the Dynamic Creation of Processes in MPI-2", Proceedings of the Conference: Recent Advances in Parallel Virtual Machine and Message Passing Interface, 13th European PVM/MPI User's Group Meeting, Bonn, Germany, September 17-20, 2006.

[59] M. Zaharia, M. Chowdhury, M. J Franklin, ScoShenker, I. Stoica,"Spark: cluster computing with working sets", HotCloud 10 (2010), 2010.

[60] Maui Administrator Guide, http://docs.adaptivecomputing.com/maui/mauiadmin.php, Accessed: 2016-01-29. www.zuj.edu.jo/conferences/ICIT05/2005/Database/123.pdf

[61] Maui Cluster Scheduler: http://www.adaptivecomputing.com/products/opensource/maui/, January 2014

[62] Moab hpc basic edition. http://www.adaptivecomputing.com/products/hpc-products/ moab-hpc-basic-edition/. Accessed: 2016-01-29.

[63] N. Goel, R.B. Garg, "A Comparative Study of CPU Scheduling Algorithms", Internal Journal of Graphics & Image Processing, Vol. 2, Issue 4, November

2012.

[64] N. Jain, A. Bhatele, X. Ni, T. Gamblin, L. V. Kale,"Partitioning low-diameter networks to eliminate inter-job interference", In Proceedings of the IEEE International Parallel & Distributed Processing Symposium, IPDPS, May 2017.

[65] N. Jain, E. Bohm, E. Mikida, S. Mandal, M. Kim, P. Jindal, Q. Li, S. Ismail-Beigi, G. Martyna, L. Kalé,"Openatom: Scalable ab-initio molecular dynamics with diverse capabilities", In International Supercomputing Conference, ISC HPC '16, 107, 2016.

[66] NERSC's Edison, Top 500, https://www.top500.org/system/178443.

[67] Open mpi, https://www.open-mpi.org/.

[68] Ostberg, P-O. , Espling, D., Elmroth, E." Decentralized scalable fairshare scheduling", Future Generation Computer Systems - The International Journal of Grid Computing and eScience 29 (2013) 130–143, 2013.

[69] Ostberg, P-O. and Elmroth, E.: Decentralized prioritization-based management ¨ systems for distributed computing. In: eScience (eScience), 2013 IEEE 9th International Conference on, IEEE (2013) 228–237, 2013.

[70] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, J. Krishna, E. Lusk, R. Thakur." Pmi: A scalable parallel process-management interface for extreme-scale systems", In Proceedings of the 17th European MPI Users' Group Meeting Conference on Recent Advances in the Message Passing Interface, EuroMPI, 2010.

[71] P. Ezzatti, M. Pedemonte, A. Martin, "An efficient implementation of the Min-Min heuristic", Eisevier, Computers & Operations Research, Volume 40, Issue 11, 2670-2676, November 2013.

[72] P. Miller, M. Robson, B. El-Masri, R. Barman, G. Zheng,A. Jain, L. Kalé,"Scaling the isam land surface model through parallelization of inter-component data transfer", In 2014 43rd International Conference on Parallel Processing, pages 422–431, September 2014.

[73] PlatformLSF,http://www.03.ibm.com/systems/platformcomputing/products/lsf/. Accessed: 2016-01.

[74] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, B. Park, H. Song, "Parsec: A Parallel Simulation Environment for Complex Systems",

Vol. 31(10), IEEE Computer, October 1998.

[75] R. Buyya (editor), High Performance Cluster Computing: Architectures and Systems, Volume 1, Prentice Hall,USA, 1999.

[76] R. Buyya, D. Abramson, J. Giddy, " Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid ", Proceedings of the 4th International Conference and Exhibition on High Performance Computing in Asia-Pacific Region (HPC ASIA 2000), May 14-17, 2000, Beijing, China, IEEE CS Press, USA, 2000.

[77] R. Buyya, H. Stockinger, J. Giddy, D. Abramson," Economic Models for Management of Resources in Peer-to-Peer and Grid Computing", SPIE International Conference on Commercial Applications for High-Performance Computing, August 20-24, 2001, Denver, USA.

[78] R. Buyya, M. Murshed,"GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing", https://arxiv.org/pdf/cs/0203019.

[79] R. Umar, A. Pujiyanta," Development of first come First Serve-Ejecting based Dynamic Scheduling (FCFS-EDS) simulation scheduling method for MPI job in a grid system", ARPN Journal of Engineering and Applied Sciences 12(8):1972-1978, January 2017.

[80] S. Asghar, E. Aubanel, D. Bremner, " A Dynamic Moldable Job Scheduling Based Parallel SAT Solver", Parallel Processing (ICPP), 2013 42nd International Conference on, pages 110–119, October 2013.

[81] S. Ghafoor, T. Haupt, I. Banicescu, R. Carino, N. Ammari, " A Resource Management System for Adaptive Parallel Applications in Cluster Environments", In Proceedings of the 6th International Conference on Linux Clusters: The HPC Revolution 2005, April 2005.

[82] S. Herbein, "Advanced Schedulers For Next-Generation HPC Systems", 2018.

[83] S. S. Vadhiyar , J. J. Dongarra," SRS - A Framework for Developing Malleable and Migratable Parallel Applications for Distributed Systems", In: Parallel Processing Letters. Volume, 2002.

[84] S. Srinivasan, S. Krishnamoorthy, P. Sadayappan, "A robust scheduling technology for moldable scheduling of parallel jobs", Cluster Computing,

2003. Proceedings, 2003 IEEE International Conference on, pages 92–99, December 2003.

[85] S. Srinivasan, V. Subramani, R. Kettimuthu, P. Holenarsipur, P. Sadayappan, " Effective Selection of Partition Sizes for Moldable Scheduling of Parallel Jobs", High Performance Computing (HiPC) 2002, volume 2552 of Lecture Notes in Computer Science, pages 174–183, Springer Berlin Heidelberg, 2002.

[86] S. Sundarensan, S. D. Johnson, V. M. Bharathy, P. M. P. Kumar, "Machine learning and IoT-based smart farming for enhancing the crop yield", Journal of Physics Conference Series 2466(1):012028, March 2023.

[87] S. Wu, Q. Tuo, H. Jin, C. Yan, Q. Weng, "HRF: A Resource Allocation Scheme for Moldable Jobs", In Proceedings of the 12th ACM International Conference on Computing Frontiers, CF '15, pages 17:1–17:8, New York, NY, USA, 2015.

[88] T.E. Carroll , D. Grosu, " Incentive Compatible Online Scheduling of Malleable Parallel Jobs with Individual Deadlines", In 39th International Conference on Parallel Processing (ICPP), 2010.

[89] W. Cirne , F. Berman, "Using moldability to improve the performance of supercomputer jobs", Journal of Parallel and Distributed Computing, 62(10):1571–1601, 2002.

[90] Y. A. Adekunle, Z. O. Ogunwobi, A.S. Jerry, B. T. Efuwape, S. Ebiesuwa, J. Ainam, "A Comparative Study of Scheduling Algorithms for Multiprogramming in Real-Time Systems", International Journal of Innovation and Scientific Research ISSN 2351-8014, Vol.12, No.1, 180-185, 2014.

[91] Y. Fan, Z. Lan, P. Rich, W. Allcock, M. E. Papka, B. Austin, D. Paul, "Scheduling Beyond CPUs for HPC", HPDC '19: Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing, pp 97-108, June 2019.

[92] https://www.cs.huji.ac.il/labs/parallel/workload/logs.html

[93] Yoo, A., Jette, M., Grondona, M." SLURM: Simple Linux Utility for Resource Management", In Feitelson, D., Rudolph, L., Schwiegelshohn, U., eds.: Job Scheduling Strategies for Parallel Processing. Volume 2862 of

Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2003) 44–60, 2003.