



Ca' Foscari
University
of Venice

Ph.D. Degree
in Computer Science

Cycle XXXVII

Final Thesis

**Optimizing Cloud Computing Performance:
Simulation of Finite-Capacity Multi-Server
Queueing Systems**

Supervisor

Professor Andrea Marin

Co-Supervisor

Professor Leonardo Maccari

Graduand

Muhammad Waqas

Matriculation number: 956673

Academic Year 2024/2025

Abstract

The concept of cloud computing has emerged from the distributed software architecture with the aim of providing services over the Internet. It has revolutionized the access and use of computational resources with remarkable flexibility and scalability. However, the dynamic nature of cloud environments, intensified by the growing demands of Artificial Intelligence-driven workloads that require a vast number of resources and capabilities, introduces serious challenges in resource allocation, service quality assurance, and performance optimization. This work addresses the challenges of resource allocation due to diverse resource types (e.g. memory, cores, bandwidth, input/output) by focusing on the simulation and analysis of finite-capacity multi-server job queueing systems within cloud computing environments, using the OMNeT++ framework. Specifically, we investigate the impact of scheduling policies on the performance metrics of a multi-server job system to contribute to a deeper understanding and provide solutions to optimize cloud resource allocation due to diverse resource types required by jobs from different types of sources.

The first part of this thesis involved developing a detailed simulation library for implementing multi-server job queueing systems. This work focused on the practical aspects of simulating cloud environments with finite resources, specifically developing an open-source simulator to analyze queueing models on the top of OMNeT++. By configuring various queueing disciplines such as First-In, First-Out and Priority-based scheduling algorithms, we were able to dynamically model resource allocation, simulate job scheduling, and analyze performance metrics, providing a robust tool for examining real-world scenarios in cloud computing. The importance of the simulator is justified by the lack of general analytical results that can predict the performance metrics efficiently.

In the second part, we extended our work by conducting a detailed simulation study of finite-capacity multi-server job systems. This study focused on understanding the impact of different queueing policies on performance metrics in the cloud environment. Our findings revealed that assigning priority to jobs with longer service times and higher resource demands can positively impact both smaller jobs with shorter

service times and big jobs, and thus enhance overall system performance.

As a follow-up to this thesis, we further investigate the performance implications of scheduling policies, particularly preemptive and non-preemptive approaches. The study demonstrated that preemptive policies, including a greedy approach in which resources are freed by first preempting smaller jobs and gradually progressing to big ones until the required resource threshold is met, improved performance metrics such as the probability of dropping, waiting time and resource utilization. However, these policies also introduce computational waste as the active jobs are disrupted during their service in this model. This trade-off emphasizes the heterogeneity of optimizing cloud computing environments and the need for careful consideration of both performance and resource efficiency.

In general, our work contributes to a better understanding of the relationship between queueing disciplines, resource allocation, and system performance in cloud computing environments. These research findings are of great help not only to cloud service providers, but also to cloud consumers for the optimal design of cloud-based systems, which is achieved by simulation tools developed during the study and the analytical insight from the experiments conducted as part of the work.

Contents

1	Introduction	1
1.1	Motivation	3
1.2	Research Contribution	3
1.3	Thesis Organization	6
2	Theoretical Background	8
2.1	Introduction	8
2.2	Foundations of Queueing Theory	8
2.2.1	Introduction to Queueing Theory	9
2.2.2	Basic Components of a Queueing System	10
2.2.3	Classification of Queueing Models	13
2.2.4	Performance Metrics in the Queueing Theory	15
2.2.5	Analytical Methods in Queueing Theory	20
2.2.6	Challenges and Future Directions in Queueing Theory	21
2.3	Scheduling Policy	22
2.3.1	Introduction to Scheduling Policy	22
2.3.2	Classification of Scheduling Policies	23
2.3.3	Preemptive vs. Non-Preemptive Scheduling	24
2.3.4	Work-Conserving Scheduling Discipline	25
2.3.5	Selection of Scheduling Policies Based on Knowledge of the System	27
2.3.6	Evaluation of Scheduling Policies	27
2.3.7	Scheduling in Data Centers	28
2.3.8	Challenges and Future Directions in Scheduling Policy	29
2.4	Discrete Event Simulation	29
2.4.1	Introduction to Discrete Event Simulation	29
2.4.2	Components of a Discrete Event Simulation	30
2.4.3	Modeling and Designing a Discrete Event Simulation	32

2.4.4	Output Analysis	32
2.4.5	Common Simulation Models	32
2.4.6	Simulation Software and Tools	33
2.4.7	Challenges and Future Directions in Discrete Event Simulation	33
2.4.8	Conclusion	34
3	Review of Scheduling Policies in Data Centers	35
3.1	Introduction	35
3.2	Literature Review on Scheduling Policies	36
3.2.1	FIFO Scheduling Policies	36
3.2.2	Priority-Based Scheduling Policies	38
3.2.3	Preemptive and Non-Preemptive Scheduling Disciplines	38
3.2.4	Evolutionary Algorithm-Based Scheduling	40
3.2.5	Energy-Efficient Scheduling	41
3.2.6	Machine Learning-Based Scheduling	41
3.3	Conclusion	42
4	Simulation Framework and Implementation Using OMNeT++	43
4.1	Introduction	43
4.1.1	Motivation and Objectives	44
4.1.2	Simulation Framework	44
4.2	The Simulation Model	48
4.2.1	Model Configuration	50
4.2.2	Model Implementation in Network Description Language (NED)	53
4.2.3	Model Implementation in C++	55
4.2.4	Scheduling Discipline	58
4.2.5	Simulation Model	59
4.2.6	Performance Metrics	61
4.3	Simulator Availability	68
4.4	Conclusion	68
5	Simulation Study of Finite Capacity Multi-Server Job Systems	69
5.1	Introduction	69
5.2	Background and Motivation	70
5.3	Queueing Model Description	71
5.4	Scheduling Policies Evaluated	72
5.4.1	FIFO Scheduler	72
5.4.2	Hard Priority Scheduler	73

5.5	Simulation Model and Implementation	74
5.6	Simulation Results and Insights	75
5.7	Conclusion	81
6	Scaling Multi-Server Job Systems: A Dynamic Approach with 29	
	Job Sources	82
6.1	Introduction	82
6.2	Motivation	82
6.3	Dynamic Queueing Model Description	83
6.4	Scheduling Policies Evaluated	83
6.5	Simulation Model Evaluation	83
6.6	Results	84
6.7	Conclusion	88
7	Performance Evaluation of Multi-Server Job System with Preemp-	
	tion and Restart	89
7.1	Introduction	89
7.2	Background and Motivation	90
7.3	Model Description	91
7.4	Scheduling Policies Evaluated	92
	7.4.1 FIFO	92
	7.4.2 Greedy FIFO	92
7.5	Simulation Model and Implementation	98
7.6	Simulation Results and Insights	100
7.7	Simulator Availability	107
7.8	Future Work	107
7.9	Conclusion	108
8	Conclusion and Future Work	109
8.1	Introduction	109
8.2	Summary of Important Findings	109
8.3	Future Work	110
8.4	Conclusion	111
9	CellA Dataset from Google Borg	113
9.1	CellA Dataset	113
	Bibliography	117

Acknowledgment

First of all, I am deeply thankful to Allah, the Almighty, for His boundless blessings, guidance, and wisdom. It is with profound regard that I acknowledge my utter dependence on His will. He is the source of all knowledge and strength, and it is through His gracious will that I have been able to undertake and finish this study.

I also wish to express my sincere admiration and respect for Hazrat Muhammad (PBUH), the last Prophet of Allah, whose teachings continue to guide and inspire countless lives, including my own. His life, full of wisdom, kindness and resilience, has been a source of strength for me during this journey. He (PBUH), His Family (R.A), and His Followers (R.A) remain a continuous source of guidance for me.

I extend my deep gratitude to my supervisors, Professor Andrea Marin and Professor Leonardo Maccari, for their tremendous support and guidance throughout this journey. Their patience, expertise, and insightful feedback have been essential in shaping this research. I am wholeheartedly thankful for their unwavering support. I would like to extend my sincere thanks to Professor Paolo Castagno from the University of Turin and Professor Simone Morosi from the University of Florence, who kindly took the time to review my thesis, highlighted key issues, and offered valuable suggestions.

My time at the ACADIA Lab in the DAIS, has been a nourishing experience, largely thanks to my lab-mates, they were helpful every time, even during all the formalities before I arrived in Venice. I am especially grateful to Dr. Diletta Olliaro, affectionately known as “CAPA” in the ACADIA. Her constant support, whether it was discussing research-related matters, navigating official tasks at the campus, or helping me deal with the complexities of Italian bureaucracy, has been invaluable. When acknowledging the officials at DAIS, one cannot forget the services and kindness of Nicola Miotello. He is a man of diverse skills and knows how to handle the problems faced by students. His support has been a great help throughout my time here.

To my beloved parents, Muhammad Fayyaz & Razia Bibi, my wife, Gul, my son, Khog Khan, and my dear siblings, your continuous support and unwavering faith in me have been the base of my success. Your sacrifices, love, and prayers have provided me with the strength and motivation to remain consistent through every challenge.

I am deeply indebted to my in-laws for their support, especially my father-in-law, Dr. Mati Ullah Khan. His family took exceptional care of my family in Pakistan during my time away for this Ph.D. in Italy. Their love, support and sacrifices have

been immeasurable, allowing me to focus on my studies with peace of mind.

During my mobility period at the National University of Science and Technology, NBC, Quetta, Pakistan, I had the privilege of working under the guidance of Professor Syed Mohsin Raza Jafri. His availability and support, even on weekends, were vital to my growth during this period. I am deeply grateful for his willingness to assist, for the skills and experience gained under his mentorship.

A special mention goes to my landlords in Mestre, Venice Sergio Boschini and Francesca, along with their family. They treated me not as a tenant, but as a member of their own family. Their kindness and generosity have made my stay a memorable and pleasant experience. They are truly one of the kindest people on earth.

I also want to express my deepest thanks to all my teachers who guided and taught me from my early school days to this thesis.

I am deeply grateful to all my friends in Italy, Pakistan, and around the globe whose guidance, motivation, and unwavering support have been a source of comfort and strength throughout my research. Your collective presence has been a beacon of hope and inspiration, pushing me to persevere even in challenging moments. Though, I may not name each of you individually, please know that your kindness and encouragement have left an indelible mark on this journey. This thesis stands as a testament to the countless ways you have lifted me up.

Lastly, I dedicate this thesis to marginalized communities all over the world and especially to people divided by the Durand Line. May this work serve as a small step toward the recognition and empowerment of those who have been overlooked and underserved.

In sum, this thesis is not merely a reflection of my work, but a testament to the collective support, guidance, and faith of all those mentioned above. To everyone who has been part of this journey, I am eternally grateful.

Chapter 1

Introduction

Rapid developments in fundamental fields such as physics, mathematics, and computer science have led to the era of the Internet followed by the explosion of big data and Artificial Intelligence (AI). These advancements have set the stage for distributed computing systems, a fundamental architectural paradigm supporting modern applications with intensive computational demands. The Industry 5.0 revolution and the adoption of 5G are further increasing the demand for performance, flexibility, and reliability to handle complex real-time jobs within large-scale distributed systems [1].

Consequently, cloud computing has emerged as a fundamental model, offering on-demand high-performance computing services. The National Institute of Standards and Technology (NIST) in the USA provides a standard definition of cloud computing [2]: “Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.” NIST also notes that this definition is subject to future revisions. This cloud model is composed of five key characteristics, three service models, and four deployment models. The characteristics highlighted by NIST include on-demand self-service, broad network access, rapid elasticity, resource pooling, and metered service. The basic service models include Software as a Service (SaaS), Infrastructure as a Service (IaaS), and Platform as a Service (PaaS). The cloud can be deployed as private, public, community, or a mix of two or more deployment models referred to as a hybrid deployment model. In this sense, cloud users are raised to a level of abstraction where they are hidden and relieved from the details and complexity of the hardware or software infrastructures that carry their computations. This effectively simplifies the costs

involved in developing and managing a computational infrastructure [3].

With the existing service models (SaaS, IaaS, PaaS) newer models, including Function as a Service, Database as a Service, and Disaster Recovery as a Service, continue to expand the scale and capabilities of cloud solutions. Cloud computing has become an integral infrastructure that cannot be overlooked in this industrial era of big data. However, despite its numerous perks, this rapid expansion of cloud computing has led to remarkable challenges, particularly in resource allocation and job scheduling. Inefficiencies in these aspects can lead to excessive energy consumption, prolonged job processing times, under-utilization of resources, and increased overall system burden. These issues not only deteriorate the Quality of Service (QoS) but also shorten the lifespan of underlying hardware components and contribute to higher carbon emissions [4]. Therefore, efficient management of resources and job scheduling in the complex cloud environment are crucial for optimizing performance, sustainability and reducing carbon footprints. To address this, numerous strategies have been engaged, such as migration of virtual machine, multi-stage methods where complex task are broken down into manageable jobs, and novel scheduling algorithms. Though, the underlying complexity of resource scheduling, influenced by several aspects such as time cycles, finite capacity of resources and their availability, environmental conditions, running costs and diversity of AI-driven workloads, often complicates these solutions. AI-driven workload include intensive tasks such as training machine learning models, implementing computer vision recommendation systems, executing deep learning inferences and processing large-scale data analytics. Heuristic and metaheuristic algorithms, including Deep Reinforcement Learning (DRL), have shown promise in optimizing such systems. However, they are typically data hungry and require a substantial amount of data, computational resources, and time to train their models for accuracy. In contrast, these simpler algorithms with lower time complexity, such as Longest Processing Time (LPT), First-Come, First-Served (FCFS), often referred to as First-In, First-Out (FIFO), Round Robin (RR), and Shortest Remaining Processing Time (SRPT), offer simplest and faster solutions but they are less optimal than their competitor.

Given the criticalness of complex systems, the ongoing research in this field aims to balance algorithmic optimization with practical complexity to enhance the efficiency and effectiveness of cloud computing systems. The insights gained from resource scheduling in cloud environment can also benefit other computing paradigms, such as grid computing, peer-to-peer computing, and the emerging models of edge and fog computing in distributed computing [5][6][7].

As cloud computing continues to evolve, the complexity of efficiently managing the

computational resources in system becomes increasingly critical. While, the advent of various cloud service models and advanced algorithms has improved resource utilization. However, a deep understanding of specific components within the cloud infrastructure is necessary to optimize performance of the system. One such critical component is the concept of Multi-Server Job Queueing Model (MJQM), which plays a leading role in managing and distributing computational workload across multiple servers. To get deeper into this, we must examine the principles of queueing theory and the classification of single-server and multi-server systems which are discussed in the next chapter of this thesis.

1.1 Motivation

The rapid expansion of cloud computing, driven by distributed software architecture, has transformed the way how computational resources are managed, accessed, and utilized. However, as cloud environments become increasingly dynamic, the need for compute-intensive AI-driven workloads introduces many complex challenges. These workloads demand considerable amounts of resources, such as memory, processing power, bandwidth, and storage. This presents important challenges in efficient resource allocation while maintaining QoS and optimizing overall system performance. These are the challenges that motivate this work for the simulation and analysis of finite-capacity MJQM (MJQM-F). MJQM-F further extends the MJQM by introducing the limit on resource capacity such as buffer and computational resources. The objective of this study is to investigate the impact of different scheduling disciplines on the performance metrics of the cloud computing model. Doing so contributes to a deeper understanding of resource allocation, scheduling strategies, and their optimization in modern and dynamic cloud environments. This study is characterized by the unavailability of extensive analytical tools that can precisely predict performance metrics and the need for novel solutions to balance performance metrics and resource efficiency in increasingly complex and diverse cloud systems.

1.2 Research Contribution

In this thesis, we address the challenges in performance evaluation of MJQM-F from different angles with numerous scheduling disciplines.

- **Development of finite-capacity multi-server job queueing simulator**

This work focused on the development of a dedicated simulation library based on the OMNeT++ framework. This tailored library aims to analyze the cloud system with finite resource capacity and dynamic resource allocation. The simulator is equipped with a finite capacity of resources and finite buffer, meaning that only a limited number of jobs can be accommodated at any given time, with subsequent jobs being dropped when the buffer overflows. It can manage jobs from multiple sources that arrive according to a time-homogeneous, independent Poisson process, with service times that are generally distributed and class-dependent. The simulator supports class-by-class configuration, enabling modeling of various resource types, such as bandwidth, memory, storage, and cores. By configuring multiple queueing disciplines, including FCFS and priority scheduling, the simulator can dynamically model resource allocation, simulate job scheduling, and analyze important performance metrics such as drop probabilities, response time, and resource utilization. Additionally, it can also distinguish between jobs based on their resource demands and service-time requirements. Besides this, it allows for visualizing the impact of Head-of-Line-Blocking (HOL) on the performance metrics. This comprehensive OMNeT++-based simulator addresses the absence of general analytical methods for figuring performance metrics efficiently and provides a reliable platform for examining real-world cloud computing scenarios. A detailed discussion of this work can be found in Chapters 4 and 6 of this thesis.

- **Muhammad Waqas**, Andrea Marin, Muhammad Mohsin Raza Jaffri, Leonardo Maccari. Simulating Finite-Capacity Multi-Server Queues in OMNeT++.

Submitted to Simulation Modelling Practice and Theory Journal.

- **Comprehensive performance analysis of different queueing policies in finite-capacity systems**

This section of the study specifically explored the dropping probabilities in MJQM-F using a simulation approach to compare two widely applied scheduling policies: FIFO and Priority. The priority-based discipline favors larger classes (high priority) of jobs. We set up scenarios involving two job classes with distinct arrival probabilities and service times, where each job requires specific amounts of resources. The impact of these policies on job-dropping probabilities due to the finite-capacity buffer was analyzed, revealing unexpected and counterintuitive behaviors in performance metrics. For example,

the findings indicate that assigning priority to jobs with low arrival probability, longer service times, and higher resource demands can, under certain conditions, positively impact the dropping probability of both small and large jobs. Additionally, we evaluated how these scheduling policies affect other key performance evaluation metrics. This includes system occupancy and resource utilization. The simulation results showed that no single queueing policy is universally optimal for different scenarios. Although FIFO performs better under low-load conditions by minimizing context switching between job types. The priority policy performs better under moderate to heavy loads, reducing the expected response time, and improving resource utilization by better handling larger jobs. This detailed analysis provides practical guidance for optimizing resource allocation in resource-constrained cloud systems and clarifies the trade-offs involved in choosing an optimal scheduling policy. This work is thoroughly discussed in Chapter 5.

- **Muhammad Waqas**, Leonardo Maccari, Andrea Marin. Finite Capacity Multi-Server Job Systems: A Simulation Study. Presented in 38th ECMS European Conference on Modelling and Simulation (ECMS 2024).

- **Investigation of the trade-off between performance and energy consumption in large-scale multi-server scheduling with preemption and restart scheduling disciplines**

We explored the trade-offs between performance metrics and energy consumption in large-scale multi-server job scheduling using preemption and restart scheduling discipline. We have presented job queueing and scheduling policies such as FIFO with and without preemption and restart. Specifically, we explored the Greedy FIFO approach which is based on active job preemption.

In this policy, our aim is to improve resource utilization by preempting an active job from service by releasing its acquired resources if doing so results in higher overall resource utilization. Killing an active job is subject to several checks and conditions, including the comparison of the job's killing cost against the resource's idling cost. The job is then added back into the buffer, where it waits to be rescheduled. By reducing both dropping probability and the waiting time of jobs in the buffer, this policy achieves better utilization of resources.

The Greedy Approach takes this concept further by selecting a job or a subset of active jobs for preemption, based on the ascending order of acquired resources in the

active jobs list. This helps optimize resource occupation and reduce waiting time for jobs in buffer. The preempted jobs are returned to the buffer and must restart their service from the beginning, which may happen multiple times to maintain high resource utilization.

While this new policy, Greedy FIFO, significantly improves resource utilization and reduces waiting times, it results in computational waste since jobs are preempted mid-service and their progress is reset to zero. This work is implemented in a dynamic set of job sources, yielding interesting initial results. Our focus is on testing it with different input parameters, such as varying arrival rate and additional conditions such as idling cost, to establish a comprehensive understanding of the trade-offs between resource utilization, dropping probability, waiting times, and computational waste due to preemption. The methodology and findings of this work are outlined in Chapter 7.

- **Muhammad Waqas**, Andrea Marin, Leonardo Maccari. When killing jobs becomes good?

Ready for submission to the Journal of Parallel and Distributed Computing.

1.3 Thesis Organization

The rest of this thesis is organized as follows: Chapters 2 and 3 discuss the theoretical knowledge of queueing theory and a literature overview of scheduling policies employed in cloud computing. Chapter 4 provides a detailed discussion of the various components of the OMNeT++-based simulator developed for this work and presents the framework to model and simulate the finite-capacity MJQM. In contrast, Chapter 5 evaluates the performance of various queueing policies in cloud computing environments with dynamic resource allocation, focusing on job-dropping probabilities and system performance using FIFO and priority-based scheduling. Chapter 6, explores the challenges of dynamic job and resource scheduling in complex, heterogeneous cloud environments driven by AI and IoT, emphasizing the need for adaptive scheduling policies. It describes a simulation-based study using an extended OMNeT++ model to evaluate the performance of a finite-capacity queueing system with dynamic job sources and variable arrival/service rates under FIFO scheduling. Chapter 7 explores the trade-off between energy consumption and performance of multi-server queues with numerous job sources using real-world traces. In particular, this chapter examines the impact of different scheduling policies, both preemptive and non-preemptive, on key performance metrics and energy waste caused by pre-

emption of active jobs. Chapter 8 outlines directions for future work, based on the findings presented in this thesis, and concludes the study.

Chapter 2

Theoretical Background

2.1 Introduction

Queueing theory addresses one of the most frustrating experiences in life: waiting. It also deals with the theory of congestion. Queueing theory examines the relationships between the demand on a service system and the delays suffered by users requesting services from this system. Queueing theory plays an essential role in the analysis of finite-capacity systems. This analysis is based on building a mathematical model to represent the arrival process of jobs in the queueing system, the discipline by which these jobs are allowed to enter the service module, and the time it takes to serve these jobs before they finish service and release the acquired resources [8][9]. This chapter therefore deals with a review of some important theoretical concepts in queueing theory.

2.2 Foundations of Queueing Theory

Queueing theory is a mathematical discipline that has found broad applications in various fields ranging from business management to information technology. This includes telephone exchanges, supermarkets, petrol stations, and data centers [10][11]. Queueing theory models and forecast the behavior of systems that handle stochastic arrivals and service demands, making it invaluable for understanding and optimizing scenarios where resources are limited and demands are variable [12].

The origins of queueing theory can be traced back to the early 20th century, primarily due to the pioneering work of Danish mathematician Agner Krarup Erlang. Erlang, a telephone exchange engineer, was assigned a task to determine the number of circuits required to provide an appropriate level of service for telephone users.

His groundbreaking work in 1909 introduced the application of the Poisson distribution to model traffic intensity [13][14]. This distribution remains the foundation for queueing theory, especially in systems where events occur independently and randomly over time [15][16]. The study of queueing theory gained significant attention when David G. Kendall introduced a modern notation for queues that remains in use today [15]. Kendall's notation is typically represented as $A/B/m/K/N/D$, where A and B denote the interarrival and service time distributions, respectively; m and K specify the number of servers and the system capacity; and N and D represent the population size and the queue discipline, respectively.

Leonard Kleinrock, an American computer scientist and the pioneer of queueing theory, expressed the queueing system as a system where jobs arrive and request sufficient resources from the pool of finite resources. The imbalance between arrival and finite resources leads to potential queues. This formalization aids in the mathematical modeling of systems where resource allocation and requirements are unpredictable, such as telecommunication networks and cloud environments. He emphasized the conservation-of-flow principle in queueing systems, which states that the rate at which a flow increases within a system equals the difference between the rate of flow into the system and the rate of flow out. This principle is central to understanding how queues form and grow over time, laying the foundation for complex analytical models. Moreover, he simplified Kendall's notation to describe a queueing system such as $A/B/m$ where A and B denote the interarrival time distribution and the service time distribution, respectively, and m specifies the number of servers. This notation helps classify and analyze different types of queueing model, making it easier to communicate and solve complex problems in queueing theory [17].

2.2.1 Introduction to Queueing Theory

Queueing theory is fundamentally involved with the dynamics of queues, and these models are idealizations of many systems from the real-world. Queueing theory answers questions such as why do queues form? How can they be alleviated or operated? How long do they persist? It provides an analytical framework for modeling systems, particularly those in which resources are limited and jobs or tasks compete for service. This theory is not just about understanding queues, but also about designing strategies to reduce delays, optimize resource utilization, and improve performance metrics.

Consider a simple example of a web server handling incoming requests. If only one

job arrives, which means that the workload is very minimal, it uses the server's resources (CPU, I/O, Memory, etc.) and leaves the system after the service is finished without delay. However, in real-world scenarios, multiple jobs typically arrive sufficiently close in time to create a job queue as each job waits for its turn, as shown in Figure 2.1. Queueing theory helps us to understand the resulting waiting times, queue lengths, and overall system performance, which are critical for designing efficient and responsive systems [18].



Fig. 2.1: *Diagram of a queue with one customer being served and five others waiting.*

The significance of queueing theory is not limited to the theoretical domain; its applications can be traced in the real-world in various domains, including computer systems, telecommunications, and operations research. By analyzing queues in these contexts, designers and administrators can optimize system performance, improve resource utilization, and enhance user experience [19]. This framework is essential to predict behavior and improve efficiency in a variety of systems [20].

2.2.2 Basic Components of a Queueing System

Queueing systems are characterized by several elementary components that determine their behavior and performance. Understanding these components is important for analyzing the efficiency and effectiveness of a queueing system. The basic components include the arrival process, the service process, the queue capacity, the queueing discipline, and the number of resources [16]. The next section explains these in detail.

Arrival Process

The arrival process serves as the input point to the queueing model. It describes how jobs such as customers, data packets, or other tasks enter the queueing system. This pace of this process directly influences the dynamics of the queue, including its length and the waiting times experienced by the jobs arriving in the system. The

arrival process is typically characterized by the distribution of inter-arrival times, which defines the intervals between consecutive arrivals.

In most queueing models, the arrival process of jobs is stochastic, which means that arrivals occur randomly over time. The Poisson distribution is frequently used to model random arrivals, especially when each arrival is independent of the other [13][16]. The arrival rate, denoted by λ , represents the rate of job arrivals per unit of time as shown in Equation 2.1. Accurate modeling of the arrival process is important for forecasting queue behavior. It also ensures that the system can handle the incoming workload without excessive delays [20][21].

$$P(X = k) = \frac{(\lambda t)^k e^{-\lambda t}}{k!} \quad (2.1)$$

where:

- λ is the average rate of job arrivals per unit time (arrival rate),
- t is the time period under consideration,
- k is the number of arrivals,
- e refers to Euler's number, the base of the natural logarithm (approximately 2.71828).

Service Process

The service process refers to how jobs are served once they reach the queue. This process is typically characterized by the service rate, denoted by μ , which represents the number of jobs that can be served per unit of time. Equation 2.2 represents the exponential distribution commonly used to model the time between events in a Poisson process. Depending on the scheduling discipline chosen for serving the workload, the jobs might be served in any order, such as assigning resources to the job that is at the head of the waiting queue or the job having the highest priority in the buffer. Like the arrival process, the service process is often modeled using statistical distributions, with the exponential distribution widely used in scenarios where the service time is random and exhibits the memoryless property. This memoryless property implies that the service times are assumed to be independent of each other, as well as the arrival process [16][18]. These assumptions simplify the analysis and are reasonable in many real-world scenarios [20][21].

$$f(t) = \lambda e^{-\lambda t} \quad (2.2)$$

where:

- λ is the inverse of the mean service time (μ), i.e., $\lambda = \frac{1}{\mu}$,
- t is the time between two convective events (jobs).

Scheduling Discipline

This refers to the set rules or sequence of steps by which jobs waiting for service are assigned the desired resources and moved in for service. This component is crucial because it determines the order in which jobs are served, directly impacting the system's efficiency and fairness. The most common queueing discipline is First-In First-Out (FIFO), where the first entity to arrive is the first to be served.

However, other disciplines also exist, for instance, Last-In First-Out (LIFO) prioritizes the most recently arrived job. In some cases, jobs may be served in a random order. Priority queueing is another common discipline in which tasks with higher priority are served before others, regardless of their arrival time and resource requirements.

In priority scheduling systems, there are two general situations: preemptive and non-preemptive. In preemptive scheduling, a higher-priority job can interrupt a lower-priority job already in service, while in the non-preemptive approach, the higher-priority entity must wait until the current service is completed. In preemptive job scheduling, there are two handling methods for the preempted job's progress. In the resume method, the service progress of the preempted job is preserved and the job is added back to the waiting queue. Later, when the job resumes service, it continues from the point where it was interrupted. In the restart method, the preempted job's service progress is reset, meaning that when the job re-enters the service, it starts from the beginning [22]. The choice of a queue discipline significantly impacts system performance, affecting how efficiency and fairness are balanced and ensuring that high-priority tasks are appropriately managed [16][19][21].

Queue Capacity

In a finite-capacity system, the queue capacity is the maximum number of jobs that can be held within the buffer at any given time. In some queueing systems, the capacity is considered infinite, allowing the queue to grow without bound, accommodating all arriving tasks. However, many practical systems have finite queue capacities. When the queue reaches its maximum capacity, the additional number

of jobs arriving will be dropped, and this is recorded in statistics as the number of jobs dropped.

The capacity of a queue is a crucial factor in system design, affecting not only the potential for service delays but also the likelihood of customer dissatisfaction due to jobs being unable to enter the queue. Designing a queueing system involves balancing the costs associated with larger queues against the risks of blocked arrivals [20][16].

Number of Servers

The number of servers in a finite-capacity system defines how many jobs can be served simultaneously. In a single-server queue, only one job is served at a time, which can lead to significant delays if the arrival rate exceeds the service rate. On the other hand, a multi-server queue has multiple servers working in parallel, allowing for the simultaneous processing of jobs. Although this type of scenario is more complex to analyze, it offers greater flexibility in handling high arrival rates and in reducing overall waiting times, queue length, and drop probability [10][16][18].

2.2.3 Classification of Queueing Models

The taxonomy of the queueing model is based on several properties that define the behavior and applicability of these models to various real-world scenarios. Understanding the classification is important for selecting the appropriate model for the system under study. This section covers the standardized notation mechanism known as Kendall's notation, which is helpful in understanding queues.

Kendall's Notation

Queueing models can be used to describe various systems, such as telecommunications, traffic engineering, computing, and service operations. Regardless of the specific application, all queueing models share common characteristics, including the arrival pattern of jobs, service times, the number of servers, system capacity, and the queue discipline which basically deals with how jobs are processed.

To summarize these parameters, British statistician David G. Kendall introduced a notation system in 1953 that has become the standard for representing queueing models. This notation is typically represented as **A/S/c/K/N/D**, where each letter defines a specific aspect of the queueing system [23]:

- **A**: The arrival process distribution or probability distribution of job arrivals, often denoted by M (Poisson process), D (Deterministic), or G (General distribution).
- **S**: It denotes the probability distribution of the service time, M (Exponential), D (Deterministic), and G (General distribution).
- **c**: The total number of servers in the system.
- **K**: It refers to the maximum capacity of the system or the number of channels, indicating the maximum number of jobs that the system can handle. This includes those in the waiting queue and being served.
- **N**: The size of the population, or the total number of potential jobs that might enter the system.
- **D**: The scheduling discipline, which specifies the order in which jobs are served or the order in which the available resources are assigned to the jobs waiting in the buffer, such as FIFO, LIFO, or priority-based systems, etc.

In many practical situations, the most commonly used notation is $\mathbf{A/S/c}$, with **K** and **N** assumed to be infinite, and **D** typically defaulting to traditional FIFO unless otherwise specified [24][23].

General Queueing Models

The literature of queueing theory contains numerous queueing models that are considered standard. They are widely used and each model is defined by its specific configuration parameters such as arrival, service processes and etc. These models include:

- **M/M/1**: This basic model represents a single-server queue with the Poisson arrival and service processes. It is one of the simplest and most studied models in queueing theory. However, to avoid overloading, the arrival frequency must be less than the service speed of the system. The **M/M/1** queue is often used to analyze systems where the job arrival rate is random and is served one at a time, with service times following an exponential distribution.
- **M/M/c**: This model extends the **M/M/1** by introducing multiple servers (denoted by **c**). It is used for systems with parallel processing where multiple servers are executing jobs. A simple example of this can be multiple

checkout counters in a supermarket. It is particularly useful for analyzing how the number of servers affects system performance, waiting times, and queue lengths.

- **M/G/1:** This queueing model is an extension to the traditional **M/M/1** queue. It replaces the service time distribution of **M/M/1** to a general distribution (denoted by **G**), rather than being limited to an exponential distribution. It is applicable in situations where service times do not follow an exponential distribution, but the arrival process remains Poisson.
- **G/G/1:** In this model, the job arrival and the service process follow general distributions. This model is the most flexible and general, accommodating a wide variety of real-world scenarios, but it is also more complex to analyze.
- **M/D/1:** This queueing model has Poisson arrivals with deterministic service times having a single server. It is useful in scenarios where service times are consistent and predictable, such as in automated manufacturing processes.

Each of the models above refers to a specific application, and the choice of model depends on the nature of the system being analyzed. Since each of these models provides information on different behavioral patterns in the system, they help understand the trade-offs between service efficiency, queue length, and waiting times in various contexts [16][18][25].

2.2.4 Performance Metrics in the Queueing Theory

In queueing theory, performance metrics are important for evaluating the efficiency and effectiveness of a queueing system. These metrics help to understand how well a system performs under given conditions and provide insights into optimizing the performance of the system [18]. The key performance metrics include utilization, throughput, average wait time, average queue length, response time, and blocking probability.

Utilization

Utilization (ρ) is a critical metric that refers to the proportion of time that the resources of the system are occupied by workload. It is denoted by ρ and calculated as the ratio of the arrival rate λ to the service rate μ multiplied by the number of servers c as shown in Equation 2.3:

$$\rho = \frac{\lambda}{c\mu} \quad (2.3)$$

For an M/M/1 system, which is a single-server system, the utilization will be:

$$\rho = \frac{\lambda}{\mu} \quad (2.4)$$

Utilization indicates how effectively the system's resources are being used. If $\rho < 1$, the system is stable, meaning that on average, the system can handle the incoming workload. If $\rho \geq 1$, the system becomes unstable, leading to infinite queue lengths over time. For example, in an M/M/1 system, if $\rho = 0.9$, it suggests that the system is busy 90% of the time and idle 10% of the time.

Throughput

Throughput refers to the rate at which jobs are completed by the system. In a stable system, the throughput (X) is equal to (λ), since every arrival eventually leads to the completion of the service. Throughput is a direct measure of how many jobs the system can handle in a given time-period and is closely related to utilization, as shown in Equation 2.5 below.

$$X = \frac{C}{\tau} \quad (2.5)$$

Here, C denotes the total number of jobs completed by the system during time τ .

Average Queue Length

This metrics refers to the jobs waiting for resources that remain in the queue until they are assigned the required resources. The average queue length, denoted by L_q , represents the average number of jobs waiting in the queue for their turn. For an M/M/1 queue, the average queue length can be calculated using Equation 2.6:

$$L_q = \frac{\rho^2}{1 - \rho} \quad (2.6)$$

For systems with general service times, such as M/G/1, the Pollaczek-Khinchine formula [26] is used as given below Equation 2.7 :

$$L_q = \frac{\lambda^2 \sigma_s^2 + \rho^2}{2(1 - \rho)} \quad (2.7)$$

where σ_s^2 is the variance of $1/\mu$ to refer to the variability in service time. Understanding the average queue length is important as it helps to determine the factors causing congestion in the system.

Average Wait Time

The average waiting time in the queue, denoted by W_q , is the average time a job waits in the queue before the resources are assigned and it is moved into active jobs to be served. It can be derived from Little's law [16], which is given in Equation 2.8. For more details about Little's Law, readers are encouraged to the relevant section ahead.

$$W_q = \frac{L_q}{\lambda} \quad (2.8)$$

For an M/M/1 queue, the waiting time is modified as shown in Equation 2.9:

$$W_q = \frac{\rho}{\mu(1 - \rho)} \quad (2.9)$$

This metric is important for assessing customer satisfaction, as prolong waiting time of jobs in system can lead to dissatisfaction, congestion in service-oriented systems and degradation in QOS.

Response Time

The response time, also referred to as the sojourn time W , is the total time that a job spends in the system. This metric sums the time spent in the waiting queue for sufficient resources W_q become available and the time spent in the system to complete its service $\frac{1}{\mu}$. For an M/M/1 queue, it is computed as Equation 2.10:

$$W = W_q + \frac{1}{\mu} \quad (2.10)$$

Response time is directly related to resource utilization. Higher resource occupation results in exponential growth in response time. In other words, a slight increase in resource utilization can significantly increase the response time of jobs in the system. This metric is particularly important in time-critical environments such as computer systems and telecommunications.

Blocking Probability

It is the likelihood that a job will be blocked or rejected from the system due to lack of space in the buffer or due to buffer overflow. This metric is especially relevant in systems with finite-capacity, such as M/M/c/K queues, where K denotes the maximum number of tasks that can be in the system (both in service and in the waiting queue). Equation 2.11 is the Erlang B formula and is used to calculate the blocking probability [10].

$$B(E, N) = \frac{\frac{E^N}{N!}}{\sum_{k=0}^N \frac{E^k}{k!}} \quad (2.11)$$

where:

- $B(E, N)$ is the blocking probability.
- E is the traffic load in Erlangs.
- N is the number of available servers.

Whenever the buffer meets its maximum capacity, any new arriving job is dropped, regardless of its origin or resource requirements.

Poisson Arrivals See Time Averages

In queueing theory, the Poisson Arrivals See Time Averages (PASTA) property connects the state of a system with Poisson arrivals. The PASTA property states that, in a queue with Poisson arrivals, the proportion of jobs arrivals that observe the system in a particular state is equal to the proportion of time the system spends in that state. This equality of fractions is what defines the PASTA property [27].

The key requirement for PASTA to hold is that the future arrivals of jobs must be independent of the current state of the system, which is a property satisfied by the Poisson arrival process. In other words, an arrival from a Poisson process observes the system as if it were arriving at a random moment in time, without being influenced by the system's history.

Mathematically, let $X(t)$ denote the state of the system at time t , and let λ be the arrival rate. The PASTA property is expressed as shown in Equation 2.12:

$$P(\text{arriving job sees } X(t) = k) = \lim_{t \rightarrow \infty} P(X(t) = k) \quad (2.12)$$

This equation indicates that the probability that an arriving job finds the system in the state k is equal to the long-run average probability π_k that the system is in the state k as t approaches infinity ∞ .

Therefore, in any system with Poisson arrivals, the expected value of any parameter observed at the instant of a Poisson arrival is simply the long-run average of that parameter. It helps to ensure that all arriving jobs, regardless of their size or class, observe the system's state in the same way. For example, a system with big and small jobs, both big and small jobs see the same statistics on the system state upon arrival. Consequently, these jobs will share the same dropping probability.

In summary, the PASTA property provides an influential insight into queueing systems by setting that Poisson arrivals observe the system's time-averaged behavior, making it an important concept in the analysis and modeling of queues. By understanding and applying these performance metrics, system designers can optimize the operation of various queueing systems, whether in computer networks, manufacturing processes, or service industries, thus improving overall QOS and customer satisfaction [19] [28].

Little's Law

Little's Law is a elementary theorem in queueing theory that provides a relationship between three key performance metrics. The includes; the average number of jobs in the system, the average arrival rate, and the average time a job spends in the system. The are generally denoted by L , λ and W respectively [16][10]. The relationship between these metrics is expressed using Equation 2.13:

$$L = \lambda W \quad (2.13)$$

For Little's Law to hold, there are a few assumptions that need to be considered.

- **Work-Conserving System:** The system must be work-conserving, which means it should never be in an idle state when it has jobs waiting in a queue. If the queue is not empty, the servers should not waste time, but instead work on jobs. In other words, the resources must not be under-utilized.
- **No Job Dropping:** This assumption means that every job that enters the system must receive service sooner or later. This job will leave the system after completing its service.
- **Stable System:** To keep the queue as short a possible or to limit the waiting queue and not let it grow indefinitely, the(λ) must be less than the system's

maximum (μ). This condition helps to keep the system in a steady-state.

- **Finite Arrivals and Departures:** Jobs must arrive and leave the system one at a time. This means that the system is not flooded at arrival and exit.
- **Long-Run Averages:** This condition of Little's law helps to avoid initial bias or abrupt changes due to non-equilibrium state of simulation. It states that the system under study must be observed for a longer duration of time so that transient effects are minimized and averages stabilize for the results to be valid. The transient effect is the temporary imbalance that a system faces before it reaches a stable state.

These assumptions ensure the precise application of Little's Law in analyzing the performance evaluation of queueing systems.

2.2.5 Analytical Methods in Queueing Theory

Analytical methods in queueing theory refer to a set of powerful mathematical tools used to model, analyze, and forecast the behavior of queueing systems. These methods enable an accurate representation of complex systems, which leads to essential insights to optimize performance and understand the dynamics of the system. This section introduces Markov Chains and Birth-Death Processes for queueing systems.

Markov Chain

A Markov chain is a basic tool in the analysis of queueing systems, particularly systems that can be described by a set of discrete states and transitions between those states. This was introduced by A.A. Markov in 1907. A Markov chain is a stochastic process characterized by the memoryless property, meaning that the future state of the system depends only on the present state, not on the sequence of events from the past. In other words, a process that does not retain memory of the past events. This property is formally known as the Markov property [29]. Mathematically, a Markov chain is defined by a set of states $S = \{s_1, s_2, \dots, s_n\}$ and a transition probability matrix P , where each entry P_{ij} refers to the transition probability from state s_i to state s_j as shown in Equation 2.14:

$$P_{ij} = \Pr(X_{n+1} = s_j \mid X_n = s_i) \quad (2.14)$$

Birth-Death Processes

Birth-death processes are a specific type of Markov chain commonly used to model queueing systems where transitions occur only between neighboring states. In this context, “births” represent the arrivals of jobs, and “deaths” represent the completion of service and the departures of jobs from the system. The process is characterized by rates λ_n for births and μ_n for deaths, where these parameters may depend on the current number of jobs n in the system [30].

The differential-difference equations governing the state probabilities $p_n(t)$ of a Birth-Death Process are shown in Equation 2.15 below:

$$\frac{dp_n(t)}{dt} = \lambda_{n-1}p_{n-1}(t) + \mu_{n+1}p_{n+1}(t) - (\lambda_n + \mu_n)p_n(t) \quad (2.15)$$

It should be noted that in the steady state, these probabilities p_n satisfy the balance equations presented in Equation 2.16:

$$\lambda_{n-1}p_{n-1} = \mu_n p_n \quad (2.16)$$

For the simple case of the M/M/1 queue, where $\lambda_n = \lambda$ and $\mu_n = \mu$, the steady-state probabilities are mathematically presented in Equation 2.17.

$$p_n = \left(\frac{\lambda}{\mu}\right)^n p_0 \quad (2.17)$$

with p_0 determined by the normalization condition. This is given in Equation 2.18

$$p_0 = 1 - \frac{\lambda}{\mu} \quad (2.18)$$

The steady state analysis of Birth-death processes is essential for deciding long-term performance metrics such as average length of queue and waiting time of jobs in buffer, and it forms the backbone of different queueing theory models.

2.2.6 Challenges and Future Directions in Queueing Theory

With the emergence of computing environments, particularly with the expansion of data centers and cloud computing, queueing theory faces major challenges. Traditional models, such as the M/G/n queues, struggle to accurately represent the complexities of modern systems that involve large-scale parallelism, dynamic workloads, and non-stationary conditions. These environments require more sophisticated models that can adapt to the inherent variability and unpredictability of the

system. A key challenge lies in modeling multi-server job systems, where understanding performance and stability under various scheduling policies remains an open problem.

Looking ahead, the integration of queueing theory with Machine Learning (ML) represents a promising avenue for research. These techniques could be utilized to optimize scheduling decisions and predict workload patterns in real-time, thus enhancing the adaptability and efficiency of queueing systems. However, this integration requires overcoming problems related to computational constraints of queueing environments. Additionally, the scalability of queueing models must be addressed, particularly as systems continue to grow in size and complexity [31][32].

2.3 Scheduling Policy

Scheduling discipline is a critical process that aids decision making in queueing. It is employed in various industries, particularly in the manufacturing and service sectors, to allocate resources to pending jobs over specified time periods with the aim of optimizing one or more goals [33]. In the context of cloud computing and heterogeneous systems, scheduling policies play a vital role in efficient resource allocation, maintaining system stability, and ensuring optimal performance.

2.3.1 Introduction to Scheduling Policy

Scheduling policies are essential in resource management, as they determine the order of jobs and the allocation of system resources to these jobs, significantly impacting the performance and efficiency of communication networks and computing systems [34]. In cloud computing, job scheduling is crucial for optimizing resource utilization by allocating jobs based on specific requirements such as CPU, memory, network bandwidth, and priority levels. Jobs may vary in their processing requirements and deadlines, making the scheduling process complex and essential for both cloud service providers and end-users. Optimal scheduling ensures that the cloud user can effectively solve problems of diverse complexity, while cloud service providers can maximize resource utilization and service efficiency, thus maintaining a balanced and effective system [18][35].

2.3.2 Classification of Scheduling Policies

Given the distinct characteristics and practical implementation, scheduling disciplines can be categorized into several types.

1. **First-In First-Out (FIFO)** FIFO is the traditional and simplest scheduling algorithm in which incoming jobs are processed in the order they arrive into the system. This scheduling policy is easy to implement and ensures fairness, although it can lead to inefficiencies such as the convoy effect, where shorter jobs (jobs with shorter service time) are delayed behind longer ones, resulting in increased average wait times. In the literature, this convoy effect is undesirable and is known as the HOL [36][37]. In Section 3.2.1, a detailed description of FIFO is presented from the literature. Similarly, Section 5.4.1 presents the mathematical formulation of the FIFO scheduling policy in Finite Capacity Queues (MJQM-F).
2. **Shortest Job First (SJF)** This policy prioritizes jobs with the shortest service time requirements, thereby optimizing the average waiting time. However, this policy requires prior knowledge of the job service time, which may not always be available. Additionally, it can lead to the starvation of longer tasks if the frequency of arrival of shorter jobs is higher, and this causes frequent preemption of longer jobs in favor of shorter jobs [38].
3. **Round Robin (RR)** This scheduling discipline assigns a fixed quantum or a time slice, to each job in the system in a cyclic order. It means that each job in the service is served for that fixed quantum in a circular way. This method ensures fairness and is particularly useful in time-sharing systems. However, the performance of RR heavily depends on the quantum length; too short a quantum leads to excessive context switching, while too long a quantum may mimic FIFO. Selecting an optimal time slice for switching directly impacts the performance of the system [39].
4. **Priority Scheduling** This policy processes tasks based on their priority levels, ensuring that more critical tasks are handled first. This approach can be implemented in both preemptive and non-preemptive forms. Although priority scheduling ensures that important tasks are addressed promptly, it risks starving low-priority tasks, which may never get executed unless mechanisms

like aging are implemented to gradually increase their priority as their waiting time in the system increases [37]. In Section 3.2.2, a detailed literature review of Priority scheduling policy is presented. Similarly, in Section 5.4.2, the mathematical formulation used for the Priority scheduling policy in MJQM-F is presented.

5. Multi Queue Scheduling

This policy splits the jobs waiting in the buffer into multiple queues based on characteristics such as priority or job source and etc. Each queue might operate on a different scheduling algorithm. This approach is highly flexible and efficient in handling diverse workloads, but adds complexity to the scheduling process [40].

2.3.3 Preemptive vs. Non-Preemptive Scheduling

Scheduling policies can broadly be classified into two categories: preemptive and non-preemptive. In preemptive scheduling, the currently running job can be interrupted to accommodate a higher priority job. This approach allows the system to always serve the most critical job, often resulting in better resource utilization and responsiveness. However, preemptive scheduling comes with additional costs, including context-switching overhead, potential resource wastage due to restarting jobs, and increased energy consumption, particularly in power-sensitive environments [41]. In some systems, the decision of preemption is not only dependent on the priority of the job but also on some other factors such as resource availability, current performance, etc. Therefore, the preemption of an active job is subject to predefined conditions.

Preemptive scheduling disciplines can be broadly divided into two categories with respect to how they deal with preempted jobs. In the first type, the preempted job is added back into the waiting queue, its service state is preserved, and later when it resumes its service, it starts from the state where it was initially preempted. This approach is known as preemption with resume while in the other preemption type, the service state of the preempted job is reset, meaning the job loses all service progress made before the preemption. It is added to the waiting queue without any preservation of its service state. Similarly, when this job is eventually serviced again, it restarts its service from the initial state as if it were a new job [42][43]. Chapter 7 presents the preemption and restart based scheduling policy proposed in this study for performance evaluation of Multi-Server Job Systems.

On the contrary, in non-preemptive scheduling a job once started its service is run until its service is over. Although this approach is simpler and incurs no context-switching costs, it can lead to resource under utilization, particularly in systems where priorities of jobs vary significantly. Non-preemptive algorithms like FIFO and SJF offer predictability and simplicity but may not be suitable for environments requiring high responsiveness. The Figure 2.2 illustrates the classification of scheduling disciplines into two broad classes: preemptive and non-preemptive. Preemptive scheduling can be sub-categorized into “with Resume” and “with Restart”, depending upon the nature of the interrupted task that either continues from the point of preemption or starts afresh. The first type of scheduling ensures fairness and simplifies job management, but leads to higher overhead due to wasted computation and increased execution time. On the other hand, preemptive scheduling with job resume reduces wasted effort and improves efficiency, but requires additional mechanism for state saving, increasing different types of overhead. This distinction is crucial in evaluating the effectiveness of different scheduling policies in heterogeneous cloud computing environments.

In summary, the choice between scheduling algorithms, either preemptive or non-preemptive scheduling, depends on the specific system requirements and the trade-offs between system responsiveness, complexity, resource utilization, and energy constraints. Preemptive scheduling is often preferred in systems where quick response times and efficient resource use are critical, despite its higher complexity and overhead. Non-preemptive scheduling, on the other hand, offers simplicity and lower overhead, making it suitable for systems where these factors are more important than responsiveness [40].

2.3.4 Work-Conserving Scheduling Discipline

Work-conserving scheduling discipline ensures that system resources are always utilized whenever jobs are waiting in the queue. This means that whenever a job is ready to be processed and a server is available, the job will immediately be assigned the necessary resources, and its service will be scheduled. At no point will a job remain in the waiting queue if there is an available server to process it. This guarantees that resources are always utilized as long as there are jobs in the buffer. Work-conserving schedulers can be either preemptive or non-preemptive [44].

Notable features of the work-conserving scheduling are listed next.

- **Immediate Service:** The system immediately begins processing the next job as soon as the server becomes free. If there is a queue of waiting jobs,

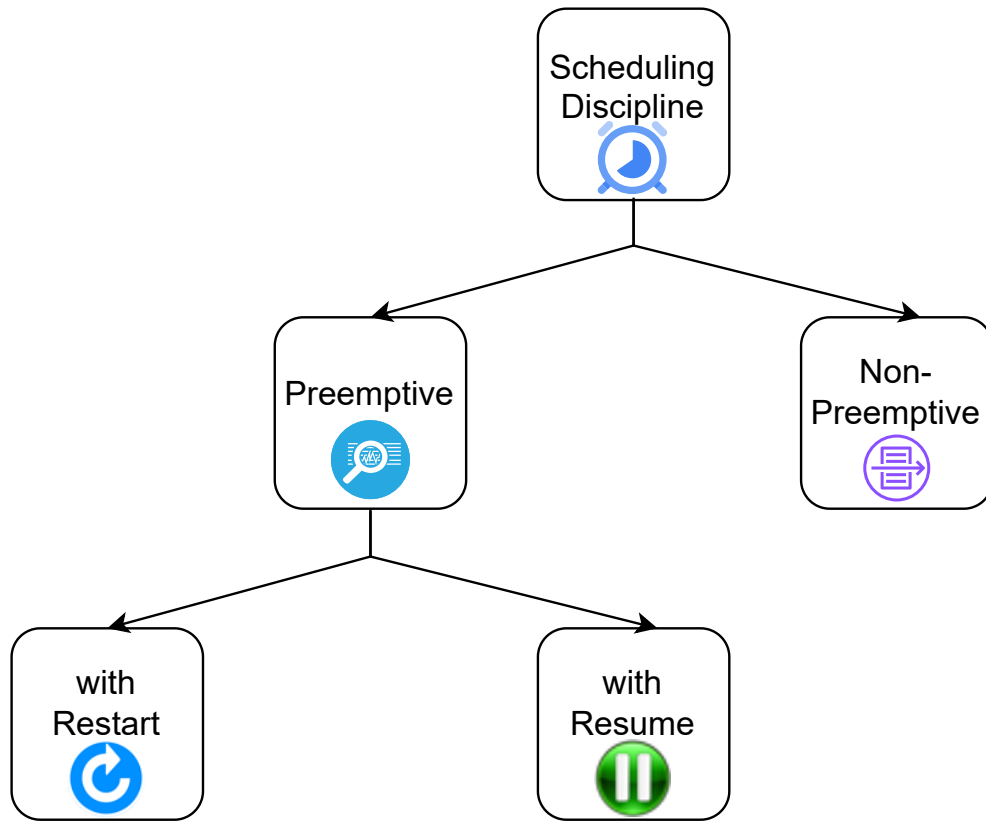


Fig. 2.2: *Overview of Preemption-Based Scheduling.*

the server will not pause or delay, thus maximizing efficiency and resource utilization.

- **No Idle Time:** The system remains occupied as long as there are jobs in the queue. The idle time only occurs when the queue is empty, indicating that there are no jobs waiting for service.

The Work-conserving scheduling disciplines maximize resource utilization and minimize the overall waiting time of jobs in the queue. These schedulers are commonly used in network traffic management, OS and other service systems where maximizing throughput is crucial.

2.3.5 Selection of Scheduling Policies Based on Knowledge of the System

A scheduling policy refers to the set of finite steps used to allocate the required resources to jobs in a computer system. These policies can vary based on the nature of the underlying system and may include priority-based approaches to ensure timely responses to critical events [45]. Selecting an optimal scheduling policy is a strategic approach. Optimal scheduling policy can significantly improve performance and optimize resource utilization. Several factors such as the type and frequency of the job the system will receive are important while making choice about scheduling policy. System capacity, such as resources and buffer also plays a vital role in this selection process. The presence of preemption or priority is another factor to consider in this selection process. Performance evaluation is directly related with the scheduling policies used. So, identification and classification of the most desired performance metrics is another key consideration. By evaluating these factors, one can select a scheduling policy that best fits the specific needs and given constraints of the system.

2.3.6 Evaluation of Scheduling Policies

Real-time scheduling theory has shown remarkable evolution in the past years. Due to the intensive work done in this area, many scheduling policies have been proposed to date [46]. The evaluation of scheduling policies is crucial to optimizing resource allocation, maintaining QoS, and improving overall system performance in systems with dynamic resource requirements and types. Evaluation statistics provide insight into how effectively a policy manages the resources of the system and optimally handles diverse workloads. The evaluation normally comes from theoretical analysis, simulation, or an actual implementation based on several key metrics which provide insight into how effectively the system resources are utilized. Common evaluation metrics include resource utilization, throughput, response time, waiting time, energy efficiency, fairness, makespan, and scalability [47]. The Section 2.2.4 discusses these metrics in detail.

Evaluating scheduling policies with the metrics stated above requires real-time analysis or detailed simulation of cloud environments with realistic traces of data. Simulators like CloudSim, OMNeT++, and other similar frameworks are used to model cloud infrastructure and workloads to observe their performance under varying conditions such as arrival rate, service rate, underlying scheduling discipline and system capacity, etc.

2.3.7 Scheduling in Data Centers

Nowadays, the cloud world is crowded with a larger number of diverse data centers which communicate with each other in an independent way offering the focused services to the users [48]. This frequent communication and workload from a diverse user base highlights the importance of scheduling and resource allocation. The resource allocation should be optimally planned to have minimize required energy to accomplish the user request while maintaining excellent QoS [49]. In the context of modern High-Performance Computing (HPC) systems, scheduling plays a crucial role, particularly as these systems evolve to accommodate the diverse demands of cloud computing. Under-utilization of data center equipment is one of the most common problems reported in the literature [50].

Traditional scheduling techniques, developed for homogeneous environments, are no longer sufficient due to the dynamic and heterogeneous nature of contemporary computing environments. Modern cloud systems are characterized by AI-driven workloads and highly variable structures, where resources can be dynamically added or removed based on workload demands. This requires advanced scheduling approaches that can monitor and adapt to real-time changes [51]. In such a system, the challenge of scheduling is compounded by the need to assign resources to jobs in a way that optimizes performance metrics such as system utilization, load balancing, and execution time. Here, jobs are often represented as nodes in a Directed Acyclic Graph (DAG), with edges denoting the communication between them. Weights are assigned to nodes and edges to reflect computation and communication costs, respectively. Scheduling can be static, where decisions are made before execution, or dynamic, where resources are allocated or reallocated during runtime [52]. This flexibility is crucial in environments where the system conditions and workloads are diverse and unpredictable [53].

The complexity of scheduling in the cloud computing environment is further illustrated by the gap between load sharing and load balancing. While load sharing aims to ensure that no resource remains idle when there are jobs waiting for execution, load balancing seeks to equalize the workload across all computing nodes, even those that are not idle. In homogeneous environments, the process migration has been an effective strategy for improving response times; however, in heterogeneous environments, this approach is less applicable due to the architectural diversity. Instead, load balancing involves a more sophisticated strategy where local schedulers not only manage resources locally, but also consider the global state of the system to optimize overall performance [54]. Chapter 3 discusses the evolution of scheduling policies for cloud computing in detail.

To summarize this discussion, scheduling in cloud environments requires addressing the problems of dynamic resource management, load balancing, and energy efficiency. Advanced scheduling techniques must not only accommodate the diverse and dynamic nature of modern systems but also optimize performance and energy consumption. Doing so can ensure that these systems can meet the demands of various modern-day applications effectively.

2.3.8 Challenges and Future Directions in Scheduling Policy

As cloud computing evolves, new challenges emerge in scheduling policies, particularly in terms of energy efficiency, dynamic resource management, and the integration of AI-driven techniques. The vast scale of cloud data centers leads to significant energy consumption, making energy-efficient scheduling critical. Virtualization technology, which enables parallel execution of tasks on Virtual Machines (VMs), must be optimized to reduce energy use while maintaining QoS standards, such as deadlines and budgets. Researchers are exploring strategies that involve dynamic QoS requirements, allowing for more flexible and adaptive scheduling, which improves resource utilization and meet application objectives more effectively [55][56].

In general, the complexity of cloud resource scheduling requires dynamic, advanced and energy efficient strategies that can adapt to the evolving demands of modern cloud environments. This includes addressing challenges such as reliability, security, and scalability to ensure that cloud services can meet the diverse needs of users while maintaining high performance and efficiency [56].

2.4 Discrete Event Simulation

Discrete-Event Simulation (DES) is one of the most well-known modeling techniques, having evolved significantly since the initiation of computer simulation in the 1950s. Specific developments in the past two decades include the utilization of AI, visual interactive modeling, simulation optimization, integration with other software, and distributed simulation [57].

2.4.1 Introduction to Discrete Event Simulation

DES is characterized by changes in the simulation state occurring at discrete time points. In other words, DES involves state changes at specific, though possibly random, intervals. Examples of systems that might be evaluated using DES include

inventory systems, manufacturing, and queueing. These systems are driven by events that occur at discrete times and change the simulation state. The event times correspond to customer arrivals and departures from a system [58][59].

2.4.2 Components of a Discrete Event Simulation

In the literature, different terminologies are used in different domains such as manufacturing, queueing, and inventory systems. However, the basic theme remains consistent. Below are the major components of DES [59][60].

1. **Events:**

This is an instantaneous occurrence that alters the state of a system. For example, in a model of a job service system, the arrival of a job would be an event that alters the system's state by introducing a new entity (the arriving job) into the model. Similarly, a job finishing its service and releasing the acquired resources is another example of an event in the DES.

2. **Entities:**

Entities are components or objects within the system that require explicit representation. These might include customers, machines or jobs. Entities interact with the system, instigating and responding to events.

3. **Attributes:**

The key characteristics of entities, such as the priority of a job, information about the resources required by a job, or the routing of a job through a manufacturing process. These attributes help to define the behavior and interactions of entities within the system.

4. **System State:**

It is a combination of variables that contain all the necessary information to characterize the system at any given time. This includes the status of entities and the values of attributes and resources within the system.

5. **Lists:**

Lists are collections of associated entities, ordered in a logical fashion, such as customers waiting in the queue for their turn. These lists help manage the flow and organization of entities within the system.

6. Future Event List:

The event list is a crucial component that keeps track of all future events, ordered by their time of arrival or any order depending on the service discipline.

7. Simulation Clock:

The simulation clock is a variable representing the current simulated time. In DES, time advances in discrete jumps from one event time to the next, as events are assumed to occur instantaneously.

8. Control Components:

Control components manage the overall flow of the simulation.

9. Random-Number Generators:

DES relies on pseudo random-number generators to produce random variables that model uncertainty in the system, such as interarrival times or service durations. These generators ensure stability in experiments. This helps to reproduce the results when the simulation is run again with similar parameter set. This is important to clone and simulate the realistic behavior of the real-world system.

10. System:

The system is the collection of different entities that interact with each other over time to meet certain objectives.

11. Model:

Abstraction hides complexity while maintaining the major functionality. Abstract representation of the underlying system is called model of that complex structure. This abstraction helps to capture the structure, logic, and relationships between different components to simulate its behavior.

12. Activities:

Activities refer to the time lap of a known length, such as service times, which are defined in advance

13. Delays:

It is the duration of unknown or indefinite length, such as waiting times in queues, where the end time is dependent on future events. In any time sensitive system, delays are always undesired.

2.4.3 Modeling and Designing a Discrete Event Simulation

The modeling and designing a DES begins with abstraction. Doing so helps to simplify a real-world system by creating a model that captures its essential elements while omitting unnecessary details and complexity. It involves identifying key entities, events, and states to represent the real-world system within the simulation environment. This highlights key operations and helps avoid inconsistencies. After the abstraction stage, validation and verification of the model is carried out. Validation ensures that the model accurately reflects the real-world system. Similarly, verification confirms that the model is correctly implemented within the simulation software. This includes checking the logical consistency of the model and verifying the proper functioning of the system [61].

2.4.4 Output Analysis

Statistical examinations of performance evaluation metrics are carried out during this stage. This aims to draw conclusions about system performance. It often includes performing the simulation iteratively with varying parameters and averaging the results to identify any trends or anomalies. Additionally, examining outliers in the data can help identify potential flaws in the system design or simulation model [61]. Generating various visual representations such as different types of plots, graphs, and heatmaps might be helpful to draw insights from the statistical results.

2.4.5 Common Simulation Models

DES commonly uses inventory models, network models, and queueing models to simulate systems. In system, entities wait in a queue for service, such as customers at a service station, jobs in a manufacturing process, and packets waiting for an acknowledgment in the network. Network models simulate the flow of information or resources between nodes, such as data packets, in a computer network. These models are essential for DES and are selected based on the specific needs and attributes of the system under study [61].

2.4.6 Simulation Software and Tools

DES is widely supported by different simulation softwares, each with distinct features and advantages. Some of the popular tools used in research and industry include NS-2, NS-3, OPNET, OMNeT++, J-Sim and QualNet. These tools are highly used in network simulation, which is crucial for modeling and analyzing the performance of network protocols without physical deployment [60][62].

NS-2 and NS-3 are both open-source, DES for networking protocols and complex systems. While NS-2 has been highly adopted due to its long history. Extensive documentation gives another edge to NS-2. NS-3 is an extended version of NS-2 for better performance, memory management, and support for scripting in Python [63]. OMNeT++ is another well-known DES with a modular architecture. This modularity makes it highly extensible and suitable for various network simulation scenarios, including wireless networks and queueing systems. OPNET is a comprehensive commercial simulator that provides extensive support for modeling and simulating the behavior and performance of networks at the packet level [64]. QualNet is a high-fidelity simulator designed to model large-scale wireless and wired networks, offering real-time performance capabilities. Finally, J-Sim is a Java-based simulation environment that supports the modeling network protocols and distributed systems [63]. These tools have the ability to model, simulate, and analyze systems in a wide range of research and engineering domains. The choice of tool is driven by several factors such as the need for open source software, ease of use, performance capabilities, or the complexity of the network being simulated.

2.4.7 Challenges and Future Directions in Discrete Event Simulation

DES faces several challenges due to the emergence of AI-driven workloads. Such workloads are very intensive and complex. These challenges include scalability, accuracy, and computational efficiency. As systems become increasingly complex, scaling DES models to simulate larger, more intricate systems without losing accuracy or efficiency is a significant problem. This problem is compounded by the need for simulations to execute within a reasonable time limit, which demand advanced computational resources and optimization techniques. Despite improvements in computational speed and reductions in modeling costs, the literature reveals a persistent gap in innovative frameworks to optimize large-scale DES models, indicating a need for more integrated and dynamic solutions [65]. Looking to the future, there is growing interest in the integration of DES with emerging technologies such

as ML and big data. This is expected to improve the accuracy and responsiveness of simulations, particularly in real-time applications, which is degraded when a large system is simulated [66].

2.4.8 Conclusion

In this chapter, we explored the theoretical ground which is crucial for developing the performance evaluation model of cloud computing through MJQM-F. We started with an in-depth discussion of queueing theory, its foundational concepts, and its importance in modeling complex systems where resource and job arrival is dynamic and diverse. These constraints lead to congestion and delays in the complex system. Furthermore, we discussed the evolution of scheduling disciplines such as FIFO, Priority, Multilevel, SJF, RR scheduling policies. We indicated their important role in resource allocation and highlighted their advantages and potential drawbacks with respect to different models. The discussion ended with the selection criteria for these policies based on knowledge of the underlying system and the desired performance metrics.

Finally, the chapter provided an overview of DES which is an important tool for modeling and analyzing the behavior of cloud systems. We discussed the important components of DES, its modeling, and design process. We also introduced various simulation software used in this domain. Overall, this chapter provides the theoretical discussion necessary to understand methodologies and challenges related to the performance evaluation of cloud computing. This helps us to set the stage for the subsequent chapters, which focus on the practical implementation and analysis of MJQM-F.

Chapter 3

Review of Scheduling Policies in Data Centers

3.1 Introduction

The rise of cloud computing and the emergence of the Internet of Things (IoT) technologies in the past two decades has dramatically transformed the way computational resources are accessed and utilized. This expansion has led to the development of massive data centers, which now support a wide range of applications including enterprise systems, AI-driven workloads, and big data analytics [67][68]. Cloud computing has shifted the traditional model of software storage and processing from client-side machines to remote data centers. In these data centers, data and services are abstracted and managed through the cloud for significant flexibility and scalability. So, users can access resources on the go through a web interface [49]. In response to the rapid advancement in IoT, cloud data centers must be capable of handling the intensive volume of data, placing new conditions on performance optimization, resource allocation, and energy consumption.

To address these problem, scheduling policies in high-performance data centers have become a critical focus of research. Efficient scheduling is important to ensure that computational jobs are optimally distributed across a large number of servers in a data center. This helps maximize resource utilization, reduce delays, and reduce energy consumption. For example, a scheduling policy that prioritizes placement of jobs on the local server will result in higher QoS and reduce network traffic. Likewise, scheduling policies that account for resource requirements, energy efficiency, and service quality can notability improve the scalability and sustainability of cloud environments [69].

This chapter offers a comprehensive examination of numerous scheduling policies in cloud data centers. It covers a wide range of scheduling policies, starting from FIFO and Priority-Based scheduling and moving to more advanced strategies such as pre-emptive and non-preemptive approaches, nature-inspired evolutionary algorithms, energy-aware, and ML-driven scheduling techniques. Through a detailed review of recent studies on scheduling policies, this chapter offers insight into their effectiveness in different cloud environments. The chapter concludes with the identification of open problems and outlines potential avenues for future research.

3.2 Literature Review on Scheduling Policies

Scheduling policies determine how resources are allocated to workloads and directly impact performance metrics such as makespan, resource utilization, energy consumption, and QoS [70]. Given the cloud systems' diverse workloads and resource types, the need for effective and adaptable scheduling strategies has grown significantly. In this work, we consider a cloud infrastructure model based on MJQM. The system consists of a set of finite-capacity computational resources and various classes of jobs. Each job class has its stochastic service time and deterministic resource demand. Jobs arrive according to a time-homogeneous, independent Poisson process, with the system using a finite-capacity buffer to manage job arrivals. If sufficient resources are unavailable, jobs must wait in the queue until the jobs that have acquired the resources earlier complete their service and release them. The simulator allows for class-by-class configuration, accommodating diverse resource types such as bandwidth, memory, and cores. By incorporating this reference model, the subsequent section provides a comprehensive review of the various scheduling policies that have been explored in the context of cloud computing.

3.2.1 FIFO Scheduling Policies

FIFO is one of the simplest scheduling disciplines in which jobs are assigned the required resources in the first-come, first-served sequence. In the study [71], the author proposed a cost-energy-efficient algorithm that adjusts the execution of jobs based on the trade-off between power and operational cost. The research highlights scheduling strategies such as FIFO, Least Running Time First (LRTF), which prioritizes jobs with the longest expected execution times to optimize resource allocation, and Least Nodes First, which allocates resources based on the jobs requiring the fewest resources, thus optimizing resource use. Their results suggest that these poli-

cies, especially LRTF and Least Nodes First, maximize the use of green energy and reduce overall costs. However, reliance on the accuracy of the prediction model and inconsistency in the availability of green energy due to fluctuations in weather and environmental conditions are the shortcomings of this approach.

The study in [72] explores scheduling techniques and resource management in cloud computing environments and large computing clusters with a focus on efficient allocation. They highlight the complexities and challenges posed by heterogeneous workloads that coexist in these environments, particularly the urgency for efficient scheduling mechanisms. In addition, the study discusses the importance of specific scheduling goals. These include distributing the workload evenly among system's resources, reducing energy consumption, and adhering to Service Level Agreements to ensure QoS. It also emphasizes that achieving an effective resource management strategy requires trade-offs between various factors. These include job makespan, energy consumption, job constraints, scalability, and cost-effectiveness.

The authors of [73] investigated the scheduling of flows in data center networks. The results of their analytical study show that scheduling delays, including communication and computational, can negate the benefits of traditional scheduling algorithms like Shortest Remaining Processing Time (SRPT). By categorizing flows into two classes (e.g, small and big), based on a certain threshold, they argue that serving small flows using FIFO without scheduling leads to lower average flow completion time. This holds even compared to SRPT under ideal conditions. The study modeled the impact of scheduling delays and showed that most small flows can benefit from non-scheduling. However, the study focuses on network flows without considering resource types and lacks discussion over the impact of how large flows might suffer under this model. The research in [74] is conducted using the traces of real-world workloads from a data center with the aim to find the most optimal scheduling strategy for energy and network cost. The study explores the cloud environment and evaluates various classical scheduling techniques, such as FIFO, SJF, and Aggressive Backfilling. Their results revealed that FCFS and Min-Min scheduling strategies are particularly effective in reducing network and energy costs.

For an overview of how we have employed FIFO in our work, readers are encouraged to refer to Section 5.4.1, which presents the characteristics of the FIFO scheduler and the mathematical formulation of the FIFO scheduling policy used to simulate the FCQ model.

3.2.2 Priority-Based Scheduling Policies

Priority-based scheduling assigns different levels of importance to jobs based on criteria such as deadlines, job size, source, and resource requirements. This allows greater flexibility; make sure that deadline sensitive sensitive jobs are completed in due time. The work done by [75] addresses energy efficiency and performance in cloud data centers. Their proposed solution focuses on reducing power consumption while ensuring that tasks meet their deadlines. This is crucial for real-time cloud services such as financial analysis, scientific computation, and flight control systems. The scheduler uses Rate Monotonic (RM) scheduling combined with a backfilling approach to fill unused processing slots to improve energy efficiency and resource utilization. The authors compare the RM scheduling method with the state of the art to highlight its superior performance in minimizing energy consumption and efficiency to meet real-time requirements. In [76], the authors propose an optimized scheduling approach using multi-objective VM selection based on measures like computational time, load volume, trust, energy usage and resource wastage. The authors introduced an entropy-based Krill Herd optimization algorithm for scheduling tasks to VMs, which outperforms the state-of-the-art such as general Krill Herd optimization, Cuckoo Search optimization, Minimum Completion Cloud, and Round Robin. The proposed model aims to improve energy efficiency, load balancing, and overall performance in cloud environments by focusing on VM selection. However, the focus of the work is limited to resource utilization and energy consumption and this somehow neglects the crucial performance metrics such as latency or response time. We employed priority scheduling and the results revealed that allocating priority to big jobs has a positive effect on small jobs as well. For an overview of how we have implemented priority in our work, readers are encouraged to refer to Section 5.4.2, which presents the characteristics of the hard priority scheduler and the mathematical formulation used to simulate the FCQ model.

3.2.3 Preemptive and Non-Preemptive Scheduling Disciplines

In the context of job scheduling, two basic approaches are widely recognized: preemptive and non-preemptive scheduling algorithms. These algorithms are crucial in managing job execution, especially in heterogeneous cloud computing environments, where resource utilization and efficiency are of greatest importance.

The work done in [77] introduces a hybrid scheduling model combining centralized and distributed approaches. It focuses on optimizing scheduling decisions by partitioning long and short jobs dynamically, using a mix of preemptive scheduling and

job-awareness techniques. Preemption is applied to avoid the HOL blocking and to better balance workloads. The solution allows for centralized scheduling of long jobs while short jobs are distributed to ensure better latency management. Since the study proposed the split of the cluster for both types (e.g. long, short) of classes, however, this could potentially under-utilize the system resources if the proportion of long or short jobs fluctuates unexpectedly. The authors in [78] introduced a novel cluster scheduling system called Lyra, to optimally handle resource (e.g., CPU, memory, bandwidth) under-utilization. The system optimizes the scheduling of jobs in GPU clusters by introducing the concept of capacity loaning to loan the idle servers to avoid or limit the idle time and elastic scaling which allows jobs to adjust the resources based on availability dynamically. Lyra also implements preemptive scheduling to handle resource reclamation efficiently. The system improves overall job queueing time, job completion time, and cluster usage, as demonstrated through large-scale simulations and a prototype implementation. Since this work focuses on the optimal allocation of GPU resources but lacks a discussion on energy consumption, the current AI-driven workload has heterogeneous resource requirements and lacks energy implications.

In [79], the authors presented a heuristic priority scheduling algorithm using the M/M/n queueing model. Their work aimed to reduce overall job waiting time in the system. A priority assignment algorithm is introduced with the help of the waiting time matrix, which assigns priorities to jobs based on their size and waiting time. However, the article focuses more on task scheduling rather than tackling the challenges of heterogeneous resource allocation and dynamic job requirements.

We propose a greedy-based preemptive scheduling policy for the performance evaluation of multi-server job systems with preemption and restart. This policy achieves higher utilization but may result in computational waste. For an overview of how we have employed this tailored approach in our work, readers are encouraged to refer to Section 7.4.2, which outlines the characteristics of the FIFO scheduler and provides the mathematical formulation of the FIFO scheduling policy used to simulate the FCQ model.

The authors in [80] proposed a model that uses non-preemptive priority queues to classify tasks into real-time and non-real-time categories, with real-time tasks given priority over non-real-time tasks. A Markovian arrival process model task arrival, which captures the interdependence of tasks. However, the need for task classification and the use of a non-preemptive priority queue with the Markovian arrival process model increase the system's complexity.

The study in [81] utilized dynamic voltage and frequency scaling to manage energy

efficiency by adjusting processor voltages and frequencies. The authors introduce a non-preemptive job discipline with deadline constraints and evaluate energy consumption and performance trade-offs in GPU-accelerated jobs. Their main goal was to limit energy consumption and maintain optimal performance up to the mark, which is critical for AI-driven workloads. Although energy-saving techniques like VM migration are beneficial for reducing energy, they may occasionally lead to service agreement violations.

3.2.4 Evolutionary Algorithm-Based Scheduling

Evolutionary algorithms, such as Genetic Algorithms and Ant Colony Optimization, are nature-aware optimization techniques. They have been employed to find near-optimal scheduling solutions in complex cloud environments. These algorithms can dynamically adapt to changes in resource demand and availability [82]. The simulation study in [83] proposed a hybrid algorithm for optimizing VM migration and job scheduling within cloud data centers. The proposed algorithm shows improved performance in energy consumption and resource utilization when compared to particle swarm optimization and ant colony optimization. Moreover, due to the complexity of the proposed algorithm, it may lead to higher execution times in large-scale environments, especially as the number of VMs increases. This can limit its scalability in extremely large data centers.

An improved version of Ant Colony Optimization for Multi-Objective Task Scheduling is proposed in [84] to optimize task scheduling by minimizing the makespan, power consumption, and load imbalance while ensuring task completion times. Despite improvements in many aspects, the proposed approach still has shortcomings with load balancing, especially as the number of tasks increases. The study acknowledges that load imbalance among VMs, especially under large work loads, is still an issue that needs further exploration.

The author in [85] proposed a genetic algorithm-based optimization algorithm. Their work claimed to efficiently balance the execution of jobs. This approach improves the dynamic adaptation of scheduling to the shifting workload demands in cloud environments. The algorithm considers multiple constraints, such as minimizing execution and waiting time while improving resource utilization. It also incorporates virtualization to provide flexible resource management and improves energy efficiency in the cloud. Similarly, given the complexity of the algorithm, it may require significant computation time for convergence, especially in real-time and large-scale heterogeneous environments.

3.2.5 Energy-Efficient Scheduling

With the emphasis on the go-green initiative, energy-sensitive scheduling policies have emerged to minimize energy consumption while maintaining system performance. These policies often trade off between resource allocation and energy efficiency, making them vital for large-scale data centers. The approach proposed in [86] integrates queueing theory to optimize server utilization by adjusting the workload in accordance with energy costs and system performance. The key concept is to minimize energy consumption by consolidating jobs into fewer servers and powering down the idle resources. Although the system optimizes for energy efficiency, it may not perform as well when dealing with highly diverse and heterogeneous workloads that have different performance requirements.

The study in [87] covers several resource provisioning models. These model ranges from static to dynamic VM consolidation, network-aware placement, and energy-aware resource management. Their survey highlights the need for adaptive, fine-grained resource management to handle short-term load fluctuations and dynamic cloud environments. However, it lacks specific simulation-based approaches.

3.2.6 Machine Learning-Based Scheduling

The integration of ML into scheduling policies allows for more intelligent and predictive resource management. ML-based scheduling can adapt to patterns in workload demand, enabling improved performance over time. The study [88] introduces a deep reinforcement learning approach using Proximal Policy Optimization to schedule tasks dynamically based on renewable energy availability. This results in a higher utilization of renewable energy. This reinforcement learning-based scheduler is designed to autonomously adapt to changes in energy availability and workload characteristics, such as unpredictable job arrivals and stochastic energy conditions. It produced valuable refinement over the state of the art. However, the study assumes that cloud jobs cannot be migrated once assigned to a server. This assumption limits its flexibility in handling real-world scenarios where jobs might need to be migrated due to failures or performance bottlenecks.

The work in [89] presents an ML approach to optimize scheduling performance in hyper-scale cloud data centers. The authors propose a gradient-boosting regression model to forecast the scheduling time for incoming jobs. They also introduced a novel resource management model that selects the best-performing resource manager dynamically. This selection is based on the predicted scheduling performance. This model is evaluated using synthetic workload traces, and it outperforms the

state-of-the-art by choosing the optimal strategy to manage resources depending on the workload and data center conditions. Although the proposed model is validated through simulation with synthetic workloads, it lacks extensive testing on real-world data. The study [90] introduces a scheduling algorithm with reinforcement learning that uses a DAG model to represent distributed tasks. The proposed algorithm incorporates a graph convolutional neural network with an attention mechanism to process scheduling information, followed by a heuristic method to allocate processors to nodes in the DAG based on the earliest finish time. Although the reinforcement learning-based algorithm improves scheduling performance, it faces challenges due to the large decision space associated with DAG scheduling, which complicates network training. In addition to this, its applicability to AI-driven cloud jobs remains unexplored.

3.3 Conclusion

This analysis has highlighted notable advancement and presented open problems in optimizing cloud computing performance, particularly in the context of finite-capacity multi-server job queueing systems. The reviewed articles explored a range of scheduling policies, including FIFO, priority-based, preemptive, non-preemptive, and ML-based scheduling approaches, each offering distinct advantages in addressing cloud resource allocation and job scheduling. These works demonstrate how diverse scheduling disciplines impact different performance metrics. However, gaps remain, particularly in addressing the trade-offs between performance optimization and resource allocation, an area that our work directly addresses by simulating realistic cloud environments using OMNeT++. By incorporating a comprehensive simulation framework, our work provides a critical contribution to understanding the complex interactions between scheduling policies, resource allocation, and system performance. This work builds on existing research while addressing limitations, particularly through its focus on balancing the competing demands of efficiency and scalability. This work highlights the importance of resource utilization and presents a balance between energy conservation and dropping probability for quality assurance in cloud environments.

Chapter 4

Simulation Framework and Implementation Using OMNeT++

4.1 Introduction

Over the past two decades, simulators have gained significant attention in various scientific domains due to their ability to replicate the behavior of complex systems. Simulating real-world events or deploying actual systems is often sluggish, expensive, technically challenging, or even unfeasible due to many constraints. However, simulations may introduce minor inaccuracies due to the inherent complexity of the systems they model, such as approximations in mathematical models or incomplete environmental variables.[91].

In the domain of communication and computer networks, where the cost of a networking components can be significant, simulators have become essential tools for both the scientific and industrial communities. Over time, network simulators have established themselves as an essential platform for testing and validating various network configurations, components, and protocols before their deployment in the real world environment. With the recent rise in investments in the cloud computing by major industrial players, the need for robust cloud simulation tools has become even more pronounced. Thus, these simulation tools are crucial for designers to test and refine their proposed solution before they are implemented in real-world cloud environments [92][93].

4.1.1 Motivation and Objectives

The study of queueing networks, particularly MJQM, has gained significant attention due to its ability to handle diverse jobs types in complex scheduling environments. Even with its potential, very little is known about MJQM in the current state of the art, particularly concerning resource wastage due to the granularity of job demands [94]. Performance analysis of such systems, especially under dynamic workloads, strongly requires accurate simulation tools for deeper analysis. In the network communication literature, several simulators have been proposed [95] in order to study the network and protocols.

In existing research, such as the work in [36][96], the authors investigated the detailed probabilistic performance of queueing networks, specifically implementing MJQM. They presented key performance indicators and their interaction with the underlying model in terms of time and cost. They acknowledge the analytical difficulty in multi-server preemptive-resume queues with general arrivals and service times. In [97], the authors explore buffer and resource allocation in multi-server queueing networks, revealing the complexities of optimizing queue configurations across dynamic workloads. Their study indicates that resource utilization and waiting times are directly impacted by resource and buffer allocations, which can be challenging to evaluate without detailed simulations. Additional studies, such as those of Lin et al. [98], examine job scheduling in heterogeneous clusters using finite-capacity buffers. The complexities involved in managing job queues in such environments underscore the limitations of current simulation tools, especially when addressing performance metrics like queue length and waiting time for multi-class job scheduling. Similarly, Kumar and Sharma [99] investigate multi-server queueing systems with discouraged arrivals and customer retention, demonstrating that the analytical complexity increases with the introduction of new parameters in the model. The importance of an accurate simulator is justified by the lack of general analytical results that can efficiently predict performance metrics.

Although prior work provides valuable insight into the intricate nature of queueing networks, they often highlight the challenges of fully capturing system performance through analytical methods alone. The next section highlights a detailed comparison of the different available simulators.

4.1.2 Simulation Framework

The modeling and simulation scene has evolved significantly in the past few decades, with simulation tools coming in and going out. This section presents an overview of

numerous commercial and non-commercial simulation tools and compares them with proposed OMNeT++ based simulation library. In addition, we explain the framework developed for this study, highlighting its relevance to our research. Considering the simulation of MJQM Scheduling, several alternative tools to OMNeT++ offer varying degrees of suitability depending on the specific requirements of the problem under study [100][101]. The following section briefly introduces the different simulators available and compares the characteristics they offer with our proposed framework (see Table 4.1 for a detailed comparison of the capabilities of each simulation tool).

1. **NS-3:**

NS-3 is a prominent discrete event network simulator widely used in academic and industrial settings [102][103]. While, primarily designed for simulating network protocols and systems, it can be adapted for job scheduling through custom modules. However, it may require significant coding overhead and customization to effectively simulate MJQM-F and multi-server scheduling. The lack of native support for granular resource allocation or dynamic job scheduling makes it less user-friendly for this specific task compared to OMNeT++ [104].

2. **CloudSim:**

CloudSim is a simulation toolkit specifically designed to model cloud infrastructure, including data centers, VMs, and scheduling disciplines [105][106]. However, its specialization in cloud computing may limit its flexibility in simulating broader network scenarios or non-standard scheduling strategies, areas where OMNeT++ might offer more comprehensive support [107][108][109]. CloudSim primarily focuses on resource allocation and virtual machine management within cloud environments. Similarly, OMNeT++ provides a modular and extensible framework for simulating detailed communication networks, distributed systems, and custom scheduling policies make it superior choice in our scenario.

3. **SimGrid:**

SimGrid is a powerful tool, particularly for large-scale distributed computing simulations. It supports the simulation of multi-server job scheduling and MJQM-F, making it a strong candidate for studies focused on performance

Table 4.1: *Comparison of Simulation Tool Capabilities*

Feature	Proposed Framework [37]	NS-3 [104]	CloudSim [108]	SimGrid [110]	Matlab [111]
Advanced Network Modeling	✓	✓		✓	
Modular System Design	✓	✓			
Cloud Resource Management			✓	✓	
Multi-Server Job Scheduling	✓	✓	✓	✓	✓
Finite-Capacity Queue implementation	✓	✓	✓	✓	✓
Ease of Customization	✓				✓
Distributed Systems Support	✓	✓	✓	✓	✓

evaluation in distributed systems [110]. However, similar to CloudSim, its primary focus on distributed computing means that it offers limited flexibility for evaluating dynamic policies like preemptive scheduling or priority-based job reordering. It may require additional effort to simulate detailed network behaviors, which are more straightforward to implement in the OMNeT++ framework.

4. Matlab:

Matlab offers a different approach with its powerful computation and modeling environment. Matlab's flexibility allows for the simulation of a wide range of systems, including multi-server job scheduling and MJQM-F, through custom scripts and Graphical User Interface (GUI) [111]. However, Matlab may not be as intuitive for simulating network-specific behaviors, which are more easily modeled in a tool like OMNeT++ lacks built-in queueing libraries for complex systems.

In summary, while each of these simulation tools has its strengths, OMNeT++ provides a fair combination of modeling network capabilities and flexibility, making it particularly suitable for simulating the scheduling of finite-capacity MJQM in diverse environments. It is one of the most accurate network simulation platforms, as it has the capability to implement and validate the performance of communication networks and scheduling strategies. It is considered a primary tool for simulating and integrating wired and wireless networks. Malik et al. [112] demonstrate the

robustness of OMNeT++ in simulating various data center operations. They describe the modeling of complex behaviors during a massive flow of network traffic. In another study, Falk et al. [113] integrate their newly proposed extension for Time-Sensitive Networks into OMNeT++ to examine and validate its performance. They ultimately extend the tool’s applications in network simulations. Similarly, Bachmeier et al. [114] discuss the capability of OMNeT++ to handle complex wired and wireless networks and validate its effectiveness. Considering the importance of queue management in complex networks, Chydzinski et al. [115] verify the stability of queueing systems in OMNeT++ by utilizing drop functions. So, they analyze the overall reliability of the queue-control strategies. The literature discussed above highlights the use and importance of OMNeT++ in network simulation. Most of these works also underline the challenges encountered when implementing models across different network simulators.

Although the previous network simulators demonstrated data center operations and arrival processes in traffic, they lack the in-depth analysis of dynamic resource allocation and the implementation of diverse job classes that are necessary to characterize modern cloud systems. Consequently, this signifies the gap between implementation and accurate simulation of MJQM-F in cloud environments. In this part of the project, we develop a standardized model of queueing networks. This development provides a more accurate and practical tool for analyzing performance metrics such as job drop probability, average waiting time, and resource utilization in modern cloud computing scenarios. The primary characteristics of our proposed tool are outlined below:

- This work demonstrates how OMNeT++, can be utilized to analyze scheduling disciplines in finite-capacity systems. Our custom-based simulator can model job queueing and resource allocation of real-world scenarios, while its design allows researchers to test and simulate different scheduling policies with minimal modifications, enhancing flexibility and reducing implementation overhead.
- We proposed a practical and feasible implementation of queueing models with finite resources, covering different non-preemptive scheduling policies. Our library provides insights into configuring simulations, defining network models, and customizing modules for dynamic cloud environments. It evaluate the performance of scheduling policies using metrics such as dropping probability, waiting time, throughput, and resource utilization. The simulator is open-source and freely downloadable, providing researchers with the opportunity to reproduce the results and explore new scheduling strategies in finite-capacity

systems.

- We presented the development of a comprehensive model for MJQM-F tailored for cloud environments. This work aimed to address the limitations of state-of-the-art network simulation models.
- The proposed model includes dynamic resource allocation and handles diverse job classes. Our model is designed to reflect these variations, providing a more realistic and flexible tool for cloud system analysis.
- Our approach employs the MJQM-F, where the arrival rate λ follows a time-homogeneous independent Poisson process, and the service time is generally distributed with a class-dependent rate μ . It also allows for class-by-class configuration of different job characteristics. This configuration allows us to perform a detailed analysis of system performance, particularly in environments with high variability in job types and server work workloads.
- This study presented the impact of scheduling policy on resource utilization. We presented how our model can be used to evaluate different scheduling strategies. This contribution not only emphasizes the adaptability of our work, but also provides reasonable insights for optimizing real-world cloud operations.

4.2 The Simulation Model

Our proposed simulation model in OMNeT++ has a modular structure, with multiple specialized modules that extend the tool's flexibility. These include the simulation kernel, environment layer and command-line interface modules, etc. The architecture is specifically based on OMNeT++ due to its general-purpose flexibility. Figure 4.1 shows the logical architecture of the proposed tool [101]. The architecture is divided into several key components. Its architecture is designed to split user interfaces, execution control, and simulation logic.

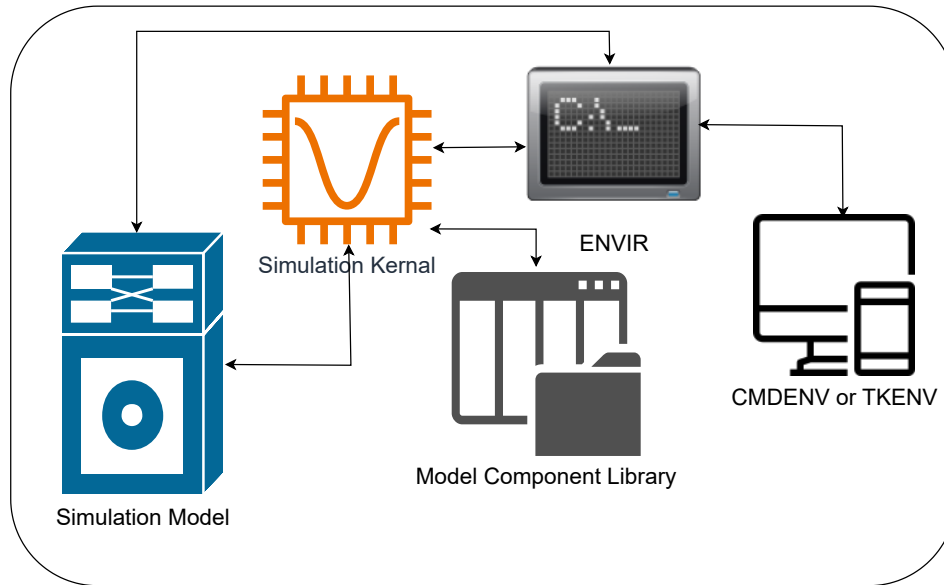


Fig. 4.1: *Logical Architecture of OMNeT++*

1. Simulation Kernel (SIM):

The kernel acts as a core component and mainly controls the execution of the simulation program. It is also responsible for performing numerous functions, including message passing, event scheduling, and execution.

2. Environment Layer (ENVIR):

The communication between the kernel and user interface is handled by this layer, which acts as a common base for various types of user interface in the tool. This layer also includes the necessary library for input and output operations and controls the debugging output. Thus, ENVIR ensures effective monitoring and control of the tool during simulations.

3. User Interfaces:

The tool provides both a command line and a graphical interface. Command Line Environment (CMDENV) has been designed to execute simulations in batch mode, while the TCL/TK Environment (TKENV) is useful for tasks such as visualizing the simulation's progress, debugging, and monitoring.

4. Simulation Model:

The kernel instantiates and manages the Simulation Model, which is the main module or process of simulation. It represents the system being simulated.

Both simple and compound components of this module are defined in the model component library.

5. Model Component Library:

In order to develop a model component library, a large collection of precompiled simple and compound modules has been devised and integrated. This repository of modular components enhances the adaptability, scalability, and capability of the tool for performing complex simulation scenarios.

In general, the proposed structure makes the tool environment powerful and adaptable to various discrete event simulation scenarios.

4.2.1 Model Configuration

The characteristics of the queueing model to implement determine the configuration of the model. To simulate the MJQM-F model, configuration parameters such as simulation time limit, number of repetitions, buffer size, and resource capacity are primarily defined in the configuration file `.ini`. The snippet mentioned in Table 4.2 refers to the generic structure of the configuration file. This file also defines the selection of the underlying queueing discipline, which will be applied during the resource allocation and job scheduling processes. The characteristics of job sources, including their inter-arrival and service times are declared. In this simulation, job arrivals are modeled as an independent Poisson process, characterized by a constant average rate of arrivals, which reflects the stochastic nature of job arrivals in a complex cloud computing environment. The inter-arrival times are exponentially distributed, reflecting the memoryless property of the Poisson process, where the arrival rate is denoted by λ (the inverse of the mean inter-arrival time). Similarly, service times follow an exponential distribution, with the service rate parameter μ defined as the inverse of the mean service time ($\mu = 1/\text{mean service time}$). In the context of queueing theory, this framework describes systems where arrivals occur randomly over time. While the current analysis assumes exponential service times (yielding an M/M/1 queue), the model can also accommodate general service-time distributions (e.g., M/G/1 queues) for broader applicability [116]. In the context of queueing theory, it describes the probability distribution of the time between consecutive job arrivals. It should be noted that the model can handle the general distribution of service time.

```

**.simTimeLimit = 500000s
repeat = 30
network = SingleQueue
**.processor.bufferSize = 512
**.processor.resourceCapacity = 256
**.processor.checkInterval = 0.01250s
**.schedulingPolicy = "FIFO" #"FIFO", "Priority"
**.source[1].interArrivalTime = exponential(2.381s)
**.source[1].serviceTime = exponential(10s)
**.source[1].requiredResource = 64
**.source[2].interArrivalTime = exponential(0.2646s)
**.source[2].serviceTime = exponential(1s)
**.source[2].requiredResource = 1
**.source[*].processor.jobDropped.record=vector
**.processor.jobProcessedSignal.record=vector

```

Table 4.2: *OMNeT++ Sample Simulation Configuration*

Figure 4.2 depicts a flowchart of the different stages involved in this simulation model. The simulation is initiated with the required configuration parameters. The model starts to generate jobs from all job sources based on the input parameters. The decision to add a job to the finite buffer or to drop it depends solely on the buffer capacity, regardless of the origin of the job or its resource requirement. Whenever a newly arrived job is dropped, the relevant counter is updated. Similarly, jobs from this finite buffer are assigned the required resources on the basis of the pre-defined scheduling policy, and relevant counters for occupied and free resources are updated accordingly. Whenever an active job finishes its service, it releases the acquired resources and is counted as a processed job for its specific source. After completion of the job, the resources are updated. In other words, in the case of any event, such as when a job acquires resources or releases them, the status of free and used resources is updated. In contrast, Figure 4.3 illustrates the queueing system with a finite-capacity buffer shared by two classes of jobs. Jobs from both sources are queued in a buffer, and jobs from each source occupy a single position in the buffer. Resource allocation or job execution occurs after a job enters the buffer. In addition, the resources are dynamically allocated based on availability. The following section elaborate on the corresponding network description (`.ned`) and the source files (`.cc`) to simulate the queueing model described above.

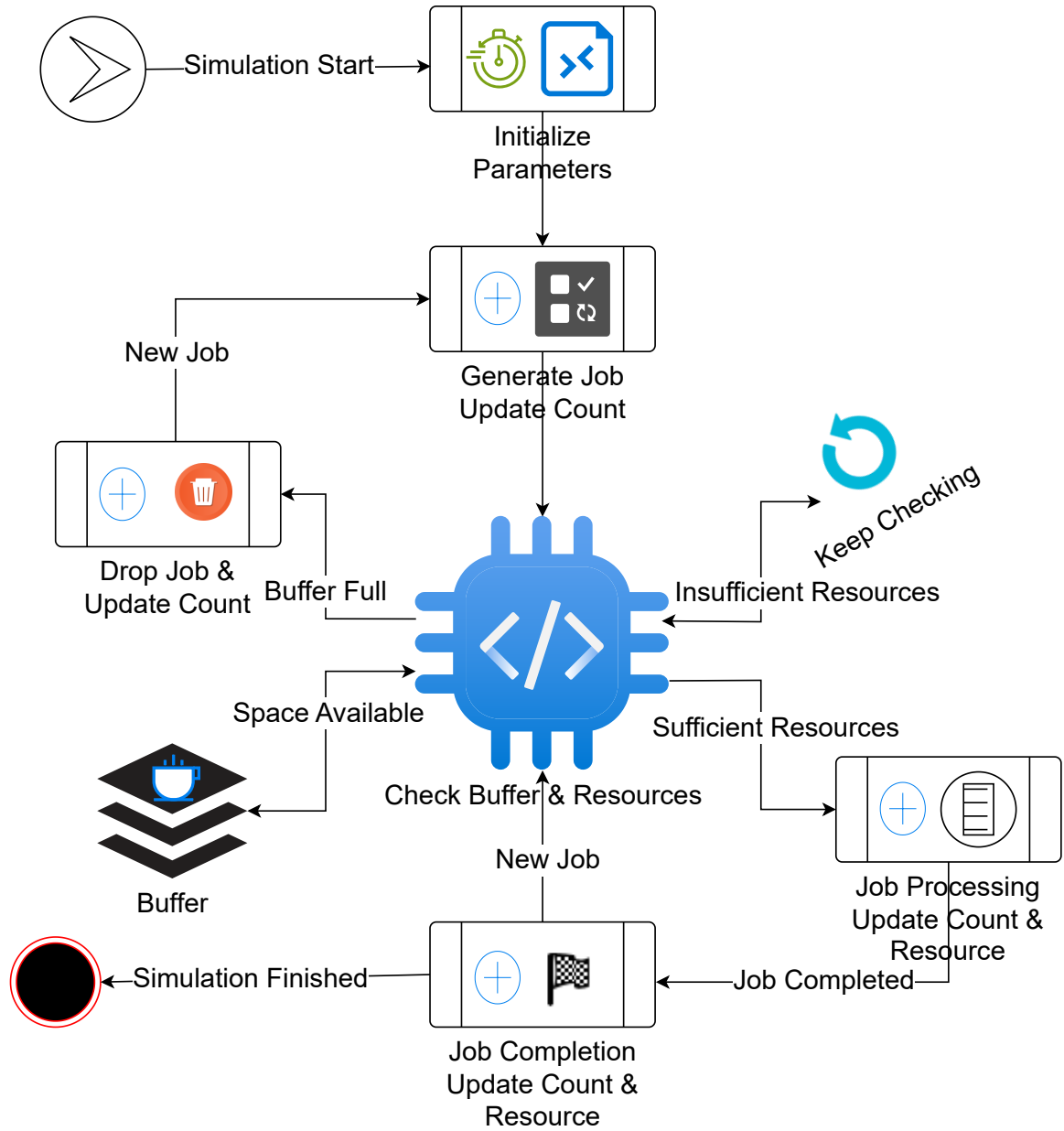


Fig. 4.2: Flowchart for Job Processing in a MJQM-F

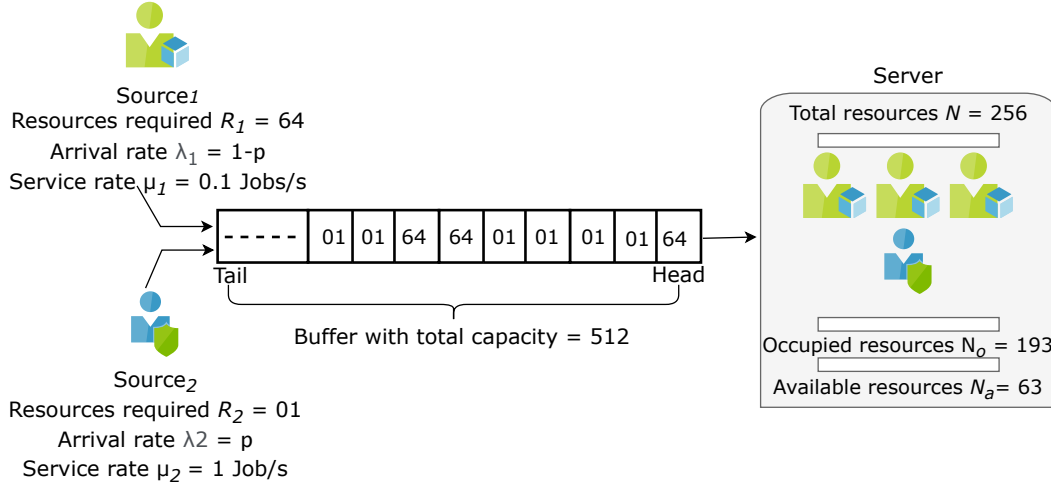


Fig. 4.3: MJQM-F Model

4.2.2 Model Implementation in Network Description Language (NED)

NED (Network Description) is a topology description language used in OMNeT++ to define the structure and interconnections of modules in a simulation model. In OMNeT++, modules represent the building blocks of a network, and NED files (.ned) are used to describe how these modules are organized and connected. In this section, we present the network configuration of our system by describing the .ned files.

Source Description

The *GenericSource.ned* (see Algorithm 1) specifies a simple module responsible for generating jobs from distinct sources at defined intervals declared in the configuration.ini file.

The SingleQueue Network Configuration establishes a pipeline for the generation, processing, and completion of new jobs in the queueing model. Jobs flow from two independent sources to a single processor. After completion of service in processor, they are moved to sink. These jobs are generated according to an independent Poisson process, commonly used in network modeling to represent random job arrivals over time [116]. The module includes inter-arrival and service time parameters, resource requirements, and statistical parameters for recording the statistics of generated jobs. This recording of statistics is later used for performance evaluation. This module serves as a job generator in the network, outputting jobs through its

out-gate.

Algorithm 1 Generic Source Module Configuration

```

1: Parameters:
2:   sourceId: string (unique identifier for the source)
3:   interarrivalTime: volatile double (time between arrivals, in seconds)
4:   serviceTime: volatile double (service time, in seconds)
5:   requiredResource: integer
6: Signals:
7:   jobGenerated: signal of type long (tracks job generation)
8: Statistics:
9:   jobGeneratedStat: records count of jobGenerated
10: Gates:
11:   out: output gate for job transmission
12: Behavior:
13: while simulation is running do
14:   wait interarrivalTime seconds
15:   create job with properties:
16:     job.sourceId = sourceId
17:     job.serviceTime = serviceTime
18:     job.requiredResource = requiredResource
19:   emit jobGenerated signal
20:   increment jobGeneratedStat counter
21:   send job to output gate out
22:   update interarrivalTime (volatile value)
23: end while

```

Queue Description

The *SingleQueue.ned* defines a network comprising two Source modules, a processor, and a sink as shown in Figure 4.4 and Algorithm 2. These modules are interconnected to form a sequential pipeline for job generation, processing, and completion. In Figure 4.4 the connections between these modules represent the flow of jobs through the system. The jobs generated by the sources are sent to the processor which is equipped with finite buffer. Whenever sufficient resources are available, jobs are assigned with required resources based on scheduling policy in place. The processor processes them and, after completion of service, the acquired resources are released and serviced jobs are sent to the sink.

Algorithm 2 SingleQueue Network Configuration

- 1: Define the **SingleQueue** network with the following submodules:
- 2: **source[2]**: An array of 2 **Source** modules.
- 3: **processor**: A **Processor** module.
- 4: **sink**: A **Sink** module.
- 5: Establish the following connections between the submodules:
- 6: Connect **source[0].out** to **processor.in**.
- 7: Connect **source[1].out** to **processor.in**.
- 8: Connect **processor.out** to **sink.in**.

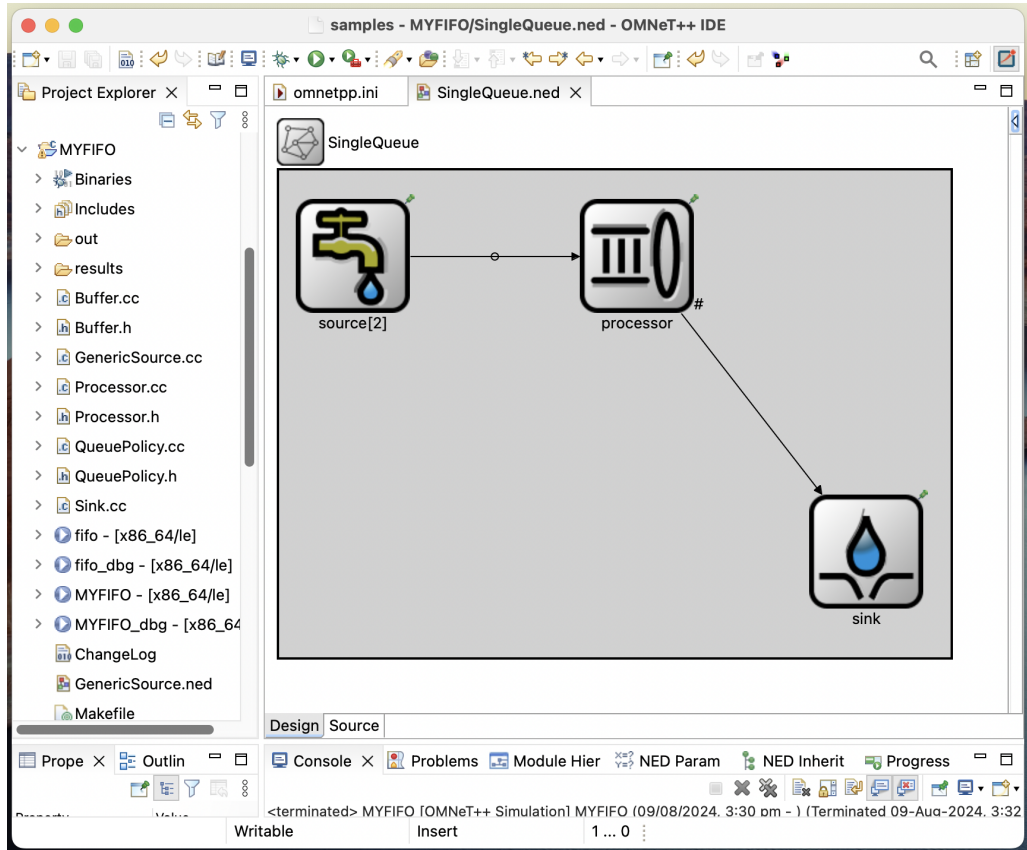


Fig. 4.4: *Design of MJQM-F in OMNeT++*

4.2.3 Model Implementation in C++

This section discusses the details of our MJQM-F implementation. The behavior and functionality of the network modules are encapsulated in various C++ source files. For the sake of simplicity, the algorithms and logic that define how the network

components interact and operate during the simulation are discussed here. For the complete implementation of the model, the reader is encouraged to refer to the accompanying GitHub repository, as detailed in the *Simulator Availability* section. Specifically, we provide an overview of the core algorithms in job generation, buffer management, queueing policy, and job scheduling, which are critical for simulating the dynamic behavior of the underlying queueing system.

Job Generation Source

This module simulates a job source that generates jobs at exponentially distributed intervals and sends them to the buffer module. It initializes the necessary parameters, schedules the first job generation event, and handles the creation and dispatch of jobs. The module uses a self-message to handle job generation timing and emit a signal to trace job generation statistics. The Algorithm 3 illustrates the core functionality of this module. Every generated job is labeled with a unique source ID and parameters based on the origin of the job.

Algorithm 3 GenericSource

- 1: **Initialize**
 - 2: Set sourceId, create sendJobEvent, schedule first event, register signal
 - 3: **handleJob**
 - 4: **If** job is sendJobEvent
 - 5: Create a job, set parameters, log details
 - 6: Send job, schedule next job generation
 - 7: Emit jobGeneratedSignal
-

Buffer Module

The buffer class is a robust implementation of a finite-capacity buffer. The buffer module handles the storage and management of incoming jobs in the system. It initializes the queue with a finite size, adds incoming jobs into it, selects the next job based on available resources and the underlying scheduling discipline (as described in Algorithm 4). Additional features of this module include a holding queue and source-based job counting which enhances its flexibility and analytical capabilities. This design makes the buffer module suitable for simulating complex systems with finite resources. The buffer module also manages information about the queue, such as the number of jobs from each class waiting in the buffer, and other similar details.

Algorithm 4 Buffer Module

```

1: Function initialize()
2:   bufferSize, queuePolicy
3:   initialize empty queue
4: Function insertJob(job)
5:   if queue length  $\geq$  bufferSize
6:     return false (buffer full)
7:   else
8:     job.arrivalTime = current time
9:     insert job into queue
10:    return true
11: Function checkNextJob(availableResource)
12:   if queue is empty
13:     return null
14:   else
15:     return queuePolicy.checkNextJob(queue, availableResource)
16: Function popNextJob()
17:   if queue is empty
18:     return null
19:   else
20:     remove and return next job from queue
21: Function removeJob(job)
22:   remove job from queue
23: Function getQueueLength()
24:   return queue length
25: Function isEmpty()
26:   return true if queue is empty, false otherwise
27: Function printQueueDetails()
28:   if queue is empty
29:     print "Queue is empty"
30:   else
31:     print details of each job in queue
32: Function getBufferCountsBySource()
33:   initialize bufferCounts as empty array
34:   for each job in queue
35:     increment bufferCounts[job.sourceId] by 1
36:   return bufferCounts

```

In addition to this, it performs checks such as determining if the queue is empty, getting the queue size, printing queue details, and counting jobs by source.

4.2.4 Scheduling Discipline

This package handles jobs from the buffer using either FIFO or Priority policy. In the FIFO policy, the available resources are allocated to jobs waiting in the buffer in their arrival order. In contrast, the Priority discipline assigns resources to high-priority (typically job with highest required resources) jobs first, which means that even if there is only one high-priority job waiting for sufficient resources in the buffer, no low-priority job can acquire the resource even though they are sufficient for this job. In case of more than one jobs having the same level of priority, the earliest arrival jobs is assigned the resources and this serve as tie breaking rule. The operation of these queue policies is detailed in Algorithms 5 and 6. The Algorithm 5 selects the first job (HOL Job) from the waiting queue if sufficient resources are available; otherwise, it returns null. If the queue is empty, no job is selected. The max-heap is used to select the job with the highest priority based on resource requirements, with ties broken by arrival time in Algorithm 6. This ensures that the most resource-demanding job is processed first if resources are available.

Efficient job handling is facilitated by resource allocation and release mechanisms. The allocation mechanisms distribute resources among jobs based on the selected scheduling policy, while the release mechanisms free resources once a job finishes its service. The released resources are re-added to the main resource pool and are assigned to subsequent jobs based on the underlying scheduling policy.

Algorithm 5 FIFO Scheduling Policy

```

1: Function PickNextJob(queue, availableResources)
2:   if queue is empty
3:     return null
4:   firstJob  $J_x$  = queue.front()
5:   if  $J_x$ .requiredResources  $\leq$  availableResources
6:     queue.dequeue() {Remove the job from the queue only if resources are al-
       located}
7:     return  $J_x$ 
8:   else
9:     return null

```

Algorithm 6 Priority Scheduling Policy

```

1: Function PickNextJob(queue, availableResources)
2:   if queue is empty
3:     return null
4:   highestPriorityJob = null
5:   maxResources = 0
6:   for each job in queue
7:     requiredResources = job.requiredResources
8:     if requiredResources > maxResources
9:       maxResources = requiredResources
10:    highestPriorityJob = job
11:   else if requiredResources == maxResources
12:     if job.arrivalTime < highestPriorityJob.arrivalTime
13:       highestPriorityJob = job
14:   if highestPriorityJob  $\neq$  null and highestPriorityJob.requiredResources  $\leq$ 
    availableResources
15:     return highestPriorityJob
16:   else
17:     return null

```

4.2.5 Simulation Model

This module manages jobs, system's resources, and computes performance metrics during the simulation. This module initializes the simulation parameters, handles job arrivals, processes jobs, and collects performance data. Detailed discussion on these metrics is found in the Performance Metrics section. The Algorithm 7 illustrates the core functionality implemented in this module, starting with setting the finite buffer, resource capacity, and initial metrics, followed by scheduling the first resource check. It handles different jobs, checks, and updates the resource capacity accordingly.

This structured approach ensures efficient job processing and resource management within the simulation. The GUI of the simulation environment is shown in Figure 4.4. This provides a structured visualization of our MJQM-F, with a seamless interface for both simulation design and analysis. On the right, the network diagram depicts the architecture of our MJQM-F with two sources generating jobs into a processor module equipped with a finite buffer and resources, followed by a sink.

Algorithm 7 Simulator Module

```

1: Function initialize()
2:   set up buffer, resource capacity, check interval
3:   initialize metrics (e.g., job count, drop count)
4:   schedule initial resource check
5: Function handleJob(job)
6:   if job is endServiceJob
7:     complete job service, update resources, process queue
8:   else if job is resourceCheck
9:     handle resource check, schedule next check
10:  else
11:    handle job arrival
12:    if buffer is not full
13:      insert job, process queue
14:    else
15:      increment dropped job count
16: Function handleJobArrival(job)
17:   set arrival time
18:   insert job into buffer or increment drop count
19:   process queue
20: Function processQueue()
21:   while buffer is not empty and can start next job
22:     pop next job, start next job
23: Function canStartNextJob()
24:   check next job
25:   if required resources  $\leq$  available resources
26:     return true
27: Function startNextJob(job)
28:   set service start time
29:   update resource capacity
30:   add job to activeJobs
31:   schedule end service
32: Function completeJobService(job)
33:   update resource capacity
34:   remove job from activeJobs
35:   send job out
36:   process queue
37: Function handleResourceCheck()
38:   schedule next resource check
39: Function finish()
40:   record final metrics

```

The sink module is used to record and collect the statistics of the served jobs. To avoid complexity, the simulation model in this study is executed with a single class (type) of resource type called cores. Furthermore, our model can be tailored to accommodate an arbitrary number of input sources, resource types, buffer capacity, and other parameters, providing users with the flexibility to simulate their specific model. This graphical layout emphasizes the flow of jobs through the system, enabling users to dynamically observe interactions between components (e.g., job generation, buffering, processing, and completion). On the left, the Project Explorer organizes the implementation files such as `.h`, `.cc`, `.ned`, and the output logs. This reflects the modular architecture of OMNeT++ and simplifies the navigation between code, configuration, and results.

4.2.6 Performance Metrics

In this section, we outline the performance metrics used to evaluate the MJQM-F. These metrics are critical to understand the effectiveness of different queueing policies in cloud computing environments. In the following section, we present some generic mathematical notations and a description of the different metrics implemented in this model is provided along with their corresponding formulas [117].

Let us introduce some generic notation:

- Let O be the set of m job sources: $O = \{O_1, O_2, \dots, O_m\}$.
- For each source O_i , let $\mathcal{J}_i = \{J_{x,i} : x = 1, 2, \dots\}$ be the set of jobs originating from O_i .
- Each source O_i has an arrival probability P_i , such that:

$$\sum_{i=1}^m P_i = 1.$$

- Each job $J_{x,i}$ (where x is the job number index and i is the source index) has the following properties:
 1. Origin: O_i (the source that generated $J_{x,i}$),
 2. Arrival time: $t_{x,i}^a$ where $t_{x,i}^a \geq 0$
 3. Service time: $\tau_{x,i}^v$ where $\tau_{x,i}^v > 0$
 4. Resource requirement vector:

$$\mathbf{r}_i = (r_i^1, r_i^2, \dots, r_i^d) \quad (4.1)$$

where each component corresponds to a different resource type (e.g., CPU cores, memory, bandwidth). A job J_i from source i waits until all of its required resources are available.

- The system has a fixed total resource capacity vector:

$$\mathbf{R} = (R^1, R^2, \dots, R^d) \quad (4.2)$$

- Let $t_{x,i}^a$, $t_{x,i}^s$ and $t_{x,i}^e$ be the arrival time, service start time, and service completion time for job $J_{x,i}$ respectively. Thus, during the interval $[t_{x,i}^s, t_{x,i}^e)$, job $J_{x,i}$ occupies resources $\mathbf{r}_i$. We call $\mathcal{S}(t)$, the set of all job ids that are in service at time t . Since, the resources are released and assigned by the scheduler, the available resource vector at time t is as shown in Equation 4.3:

$$\mathbf{R}(t) = \mathbf{R} - \sum_{(x,i) \in \mathcal{S}(t)} \mathbf{r}_i \quad (4.3)$$

where the summation is component-wise over all jobs J_i active at time t .

- FIFO: Since the FIFO scheduling policy, follows the order of arrival, let us call it:

$$t_1^a \leq t_2^a \leq \dots,$$

the arrival times of the jobs, for each job J_x from the source i . In FIFO, the start time under multi-dimensional resource constraints is defined as:

$$t_{x,i}^s = \min \{t \mid t \geq t_{x,i}^a, \mathbf{R}(t) \geq \mathbf{r}_i\} \quad (4.4)$$

where the inequality $\mathbf{R}(t) \geq \mathbf{r}_i^d$ is interpreted component-wise:

$$R^\ell(t) \geq r_i^\ell \quad \text{for all } \ell = 1, \dots, d.$$

- Priority: Here the resources are assigned on the basis of priority as elaborated in 6. We assign priority based on the amount of resources requested by the

job. For each job $J_i \in \mathcal{J}$, we define a *priority measure* π_i in Equation 4.5 as:

$$\pi_i = \sum_{\ell=1}^d r_i^\ell \quad (4.5)$$

In Priority scheduling technique, at any scheduling decision point such as a new job's arrival into the buffer or the moment resources become available:

1. Among the jobs that have arrived but not yet started, select the job with the *highest* priority π_i .
2. If two or more jobs share the same highest priority, use the earliest arrival time t_i^a as a tie-breaking rule.

No lower-priority job π_y may start if a job J_i with higher priority π_i is waiting for sufficient resources to become available such as $(\pi_y < \pi_i)$.

1. **Job Dropping Probability:** The dropping probability measures the likelihood that a job from the source i is dropped because the buffer is full, as shown in Equation 4.6. It provides insight into the system's ability to handle incoming traffic without losing jobs. In the case of buffer overflow, the incoming job is dropped without any distinction between job class, priority, or resource requirement.

$$P_i = \frac{J_{d(i)}}{J_{g(i)}} \quad (4.6)$$

Here, $J_{d(i)}$ and $J_{g(i)}$ represent the number of jobs dropped and the number of jobs generated by the source i , respectively.

2. **Average Waiting Time:** This metric indicates the average time $t_{x,i}^s - t_{x,i}^a$ that a job x from source i spends waiting in the buffer before the required resources $\mathbf{r}_i^d$ are assigned. It is crucial to analyze the delay experienced by jobs in the queueing model. Mathematically, the waiting time is computed as Equation 4.7.

$$W_i = \frac{1}{J_{p(i)}} \sum_{x=1}^{J_{p(i)}} (t_{x,i}^s - t_{x,i}^a) \quad (4.7)$$

where:

- $J_{p(i)}$: The total number of jobs processed from the source i .
- $t_{x,i}^s$: The time at which the x -th job starts execution.

- $t_{x,i}^a$: The time at which the x -th job arrives in the buffer.

3. **Average Service Time:** This time refers to the average duration a job from source i spends in execution (process). It is computed as Equation 4.8. It is an important measure of the processing efficiency of the system.

$$S_i = \frac{1}{J_{p(i)}} \sum_{x=1}^{J_{p(i)}} (t_{x,i}^e - t_{x,i}^s) \quad (4.8)$$

where:

- $t_{x,i}^e$: The time at which the x -th job releases the resources and completes its service.

4. **Average Response Time:**

This metric τ_i combines the waiting time ($t_{x,i}^s - t_{x,i}^a$) and the service time ($t_{x,i}^e - t_{x,i}^s$), representing the total time a job from the source i spends in the system as given in Equation 4.9. It reflects the overall performance from the job's perspective.

$$\tau_i = \frac{1}{J_{p(i)}} \sum_{x=1}^{J_{p(i)}} (t_{x,i}^e - t_{x,i}^a) \quad (4.9)$$

5. **Resource Utilization:**

Resource Utilization ρ measures the percentage of system resource capacity utilized during the simulation as shown in Equation 4.13. High resource utilization indicates efficient use of available resources, while under-utilization represents resource wastage. A high percentage of resource utilization is desirable as it indicates the optimal performance of the underlying model. In a complex system, the performance measures (resource utilization in this case) can be computed using several approaches. The section ahead explains the pros and cons of different methods for computing utilization with respect to DES.

- **Analytical Method:** This method is based on fundamental queueing theory principles. It computes ideal continuous-time average utilization as

shown in Equation 4.10.

$$\overline{R} = \frac{1}{T} \int_0^T R(t) dt \quad (4.10)$$

where $t_1, t_2, t_3, \dots, t_K$ are the discrete time points and T represents the total time interval.

This equation requires continuous tracking of resource occupation at every instant of time. Moreover, in DES, system events happen at discrete points so, continuously computing $R(t)$ is not practical due to the infinite resolution it demands.

- Event-based Method:

In DES, precise event-based performance evaluation can be computed using Equation 4.11. Since the event occurs are time $t_1, t_2, t_3, \dots, t_k$ where the resource occupation changes

$$\overline{R} = \frac{1}{T} \sum_{i=1}^{K-1} R(t_i) \cdot (t_{i+1} - t_i) \quad (4.11)$$

where $u_1, u_2, u_3, \dots, u_J = T$

- Sampling based Method:

In equally spaced intervals of distributed system, the resource utilization can be computed by sampling the occupation at regularly spaced sampling times as shown in Equation 4.12 below.

$$\overline{R} \simeq \frac{1}{J} \sum_{i=1}^J R(u_i) \quad (4.12)$$

where u_j refers to regularly spaced sampling times.

The system's resource usage is periodically evaluated during the simulation at specified intervals using Equation 4.12. In our simulation configuration, the check interval C_{int} was set to 0.1250s to achieve realistic performance statistics while balancing computational efficiency with measurement accuracy. While direct measurement could theoretically track utilization continuously, it would incur prohibitive overhead in event-dense simulations. Sampling aligns with real-world monitoring practices (e.g., periodic system checks) and ensures scalability. However, sampling introduces several disadvantages, such as temporal resolution issues and sensitivity to interval selection. A poorly chosen interval

could lead to either excessive computational overhead (if too small) or loss of critical details (if too large). Our selection of $C_{\text{int}} = 0.1250$ sec. was carefully determined to strike a balance between computational efficiency and sufficient resolution for capturing relevant system dynamics. In this simulation study, the resource utilization is computed using Equations 4.11 and 4.12 as they are two complementary approaches. Doing this helped us with cross-verification. We implemented a single instance of the queueing model with these two methods and observed that resource utilization converges to nearly equal values as in Table 4.3. To ensure the reliability of the utilization computed by both approaches stated above, the simulation was run for more than 2.2 million events and repeated 30 times.

Utilization Method	Value	Percentage Difference
Sampling Based Average	77.0906576	0.0295%
Event Based Approximation	77.1133729	

Table 4.3: *Comparison of Sampling-Based and Event-Based Utilization Averages with Percentage Difference*

From the table above, it is evident that the differences are quite small, indicating that both computation methods give nearly identical results in terms of resource utilization.

In systems with more than one type of resource, such as memory, bandwidth, and cores, each type of resource has its own utilization metric.

For instance, ρ_{Memory} represents the utilization of memory and ρ_{Cores} represents the utilization of cores. The utilization for each resource type d is computed separately.

$$\rho^d = \frac{\sum_{i=1}^{C_{\text{int}}} (R^d - R^d(t_i))}{C_{\text{int}} \times R^d} \times 100\% \quad (4.13)$$

where:

- ρ^d : Utilization of resource type d (e.g., ρ^{Memory} , ρ^{Cores} , ...).
- R^d : Total capacity of resource type d .
- $R^d(t)$: Available capacity of resource type d at time t .
- t_i : Sampling time of sample i .

- C_{int} : Total number of checks performed during the simulation.

6. **Number of Jobs in Service:** This metric represents the average number of jobs being processed $L_{s,i}$ simultaneously over a period from the source i as presented in Equation 4.14. It helps in understanding the system's load management capability.

$$L_{s,i} = \frac{1}{C_{\text{int}}} \sum_{x=1}^{p(i)} L_s \quad (4.14)$$

where:

- L_s : The number of jobs in service at the time t .

7. **Number of Jobs in Buffer:**

In finite-capacity systems, the buffer can accommodate a specific number of jobs. This metric $L_{b,i}$ indicates the average number of jobs L_i waiting from the source i in the buffer as presented in Equation 4.15. It provides insight into the congestion and delay within the system. This metric highlights the capability of the system to handle new jobs. If the frequency of buffer overflow is high, it will lead to a higher number of jobs dropping for each class of jobs.

$$L_{b,i} = \frac{1}{C_{\text{int}}} \sum_{x=1}^{p(i)} L_b \quad (4.15)$$

where:

- L_b : The number of jobs in the buffer at the time t from the source i .

8. **Average Throughput:** This measure refers to the average rate at which jobs are processed over a given period of time in a system given in Equation 4.16.

$$\Theta_i = \frac{J_{p(i)}}{t}, \quad (4.16)$$

where:

-
- t : The total simulation time of the system.
- Θ_i : The throughput for the source i , measured in jobs per unit of time.

These performance metrics play an essential role in evaluating and comparing different queueing policies. They provide a comprehensive understanding of the system's behavior under various conditions, aiding in the optimization of resource allocation and job scheduling strategies.

4.3 Simulator Availability

The complete project such as the source code for our models, including the FIFO and priority queueing disciplines, is publicly accessible on our GitHub repository. This repository includes detailed files such as the codebase, network descriptions, configurations, and result files. For further information and to access these resources, please visit our GitHub page.¹

4.4 Conclusion

This chapter presented a detailed discussion of the simulation framework for analyzing finite-capacity MJQM-F in cloud computing environments. The proposed OMNeT++ based simulation library successfully addresses the dynamic challenges of cloud resource management by incorporating key aspects such as multi-server job scheduling, finite-capacity queues, and scheduling policies like FIFO and priority-based disciplines. This simulator offers a modular and extensible architecture that allows for accurate modeling of diverse cloud environments, providing a valuable tool for examining the impact of different scheduling strategies on performance metrics. The versatility of the OMNeT++ framework facilitated the modeling of both network behaviors and cloud-specific scenarios, bridging a crucial gap in existing simulation tools. The performance metrics defined within this simulator offer a view of system's behavior, with an emphasis on dropping probability of jobs, average waiting and response times, resource utilization, and job distribution. By providing an open-source implementation of MJQM-F, this chapter contributes a practical and adaptable framework for both researchers and practitioners seeking to optimize cloud computing performance.

¹OMNeT++ MJQM Repository on GitHub: https://github.com/muhammadwaqas01/MJQM-F_OMNeTpp

Chapter 5

Simulation Study of Finite Capacity Multi-Server Job Systems

5.1 Introduction

For the last few decades, cloud computing has become a key part of how we use technology, offering a flexible and scalable way to access computing power [118]. However, the diverse nature of the cloud computing environment, with its wide range of resource types and capabilities, presents challenges in managing workloads and ensuring QoS. At present, the cloud computing field is mainly facing with the following problems: (a) resource scheduling, (b) task segmentation, and (c) network transmission. In addition to this, due to the variety of resources in the cloud and the different needs of various applications, finding the optimal scheduling strategy can be challenging [119][120][121]. Improving the reliability of cloud systems can be achieved by carefully analyzing and modeling key elements such as job scheduling, service time, waiting period for these services, and the likelihood of hardware and software failures [118].

Understanding the performance metrics of different queueing policies in a scalable cloud computing environment is essential for both cloud service providers and consumers [122]. In this work, we have considered a simple abstraction of a cloud infrastructure based on MJQM. Unlike other research streams in the field, we have employed the MJQM-F rather than an infinite one due to the fact that the stability condition for unbound system is not known. [123][124][125].

5.2 Background and Motivation

The constantly evolving nature of cloud computing environments, characterized by a wide variety of resource types and capabilities, presents significant challenges in managing workloads and ensuring QoS. This diversity arises from the fact that cloud resources—such as virtual machines, storage, bandwidth, and processing capabilities, are often heterogeneous in terms of their performance, scalability, and availability. Key challenges include resource service scheduling, which refers to the optimal allocation of available resources such as CPU time, bandwidth, memory, or storage to the jobs waiting for these resources to start their service; task segmentation, which entails breaking down large workloads into smaller, more manageable tasks, allowing for parallel processing; network transmission, which focuses on the efficient transfer of data packets from source to destination without loss and intact; and the application of real-world tasks. Given the diversity of resources and the varying needs of applications, finding optimal scheduling strategies is complex but crucial for improving the reliability and performance of cloud systems.

One of the most important aspects of cloud computing is the selection and implementation of queueing policies, which have a direct impact on the efficiency of the cloud environment and, consequently, the QoS experienced by end-users. In this context, understanding the performance metrics of different queueing policies in a scalable cloud computing environment is essential.

Cloud computing has transformed the way computational resources are accessed and utilized. Moreover, the dynamic nature of the cloud computing environment, which is characterized by different types of resources and capabilities, presents open problems in managing workload and ensuring QoS. The selection and implementation of queueing disciplines can have a major impact on the efficiency of the cloud environment, and thus on the service quality experienced by the end-users. Understanding the performance metrics of different queueing policies in cloud computing environments with scalable resource management is essential for both cloud service providers and consumers. In response, this work aims to evaluate the effectiveness of some queueing policies in cloud environments characterized by dynamic resource allocation with a particular emphasis on their dropping probabilities.

We propose a simulation approach that combines the development of an accurate simulation model of a cloud computing environment with adaptable resource management along with a comprehensive performance analysis of different queueing policies, including FIFO and Priority queueing. The insights from the simulation results revealed that assigning priority to jobs with longer service times and higher resource

demands can also positively impact both big and small jobs. The contributions of the proposed work are summarized as follows:

1. **Development of a Comprehensive Simulation Model:**

We developed a detailed simulation model tailored to cloud computing environments, utilizing the OMNeT++ framework [100][126]. This model is designed to accurately represent the dynamic nature of cloud resources and their allocation. To promote further research and collaboration, we have made the codebase open source and available on GitHub (See the Section 4.3 for more details).

2. **In-depth Analysis of Queueing Policies:**

Our study provides a detailed evaluation of two widely implemented scheduling disciplines: FIFO and Priority. In FIFO, the jobs arriving in the buffer are assigned with available resources in the order of their arrival, while the priority-based discipline gives precedence to jobs with larger resource demands. We focused on analyzing the dropping probabilities within finite-capacity buffers. The simulation outcome disclosed unexpected and counterintuitive behaviors in performance metrics, particularly in scenarios involving two job classes. These findings offer valuable insight into the complex interactions between queueing policies and system performance in cloud computing environments.

3. **Impact Assessment on Small Jobs:**

One of the key findings of our work is the positive impact on small jobs resulting from the precedence given to big jobs in the priority scheduling. The prioritization of jobs with longer service times and higher resource demands revealed a positive impact on the performance evaluation factors of both big and small jobs. This counter-intuitive result suggests that priority assignments in queueing can enhance resource utilization and improve service quality across different job sizes, contradicting traditional assumptions in queue management.

5.3 Queueing Model Description

One of the outstanding features of scheduling in data centers is the relatively low utilization that they can achieve under the FIFO scheduling discipline [127]. In order to investigate the impact of this phenomenon on the dropping probabilities

in finite-capacity queueing systems, we employed the MJQM-F with the following characteristics: (i) we have N identical servers; (ii) jobs arrive at the system belonging to one of two classes (sources), where the first class (*small*) requires one server, and the second class (*big*) requires a larger number of servers; (iii) jobs arrive according to a time-homogeneous independent Poisson process; (iv) jobs belong to the small class with probability p (independent of any other process) and to the other one with probability $1 - p$; (v) each job has a service time that is exponentially distributed with class-dependent rate. The job scheduler decides which job to fetch from the jobs waiting for service in the buffer whenever there are free computational resources. When the buffer reaches its full capacity, any incoming job, regardless of its source class, is dropped. In the following subsection, we describe the FIFO and the Priority scheduling policies that we compare using our simulator.

5.4 Scheduling Policies Evaluated

The above-stated queueing model is simulated with FIFO and Priority scheduling disciplines and the subsequent sections discuss the analysis of both policies with respect to the simulation model parameters.

5.4.1 FIFO Scheduler

This policy operates on a First-Come, First-Served basis, where the server allocates available resources to jobs in the buffer in the order of their arrival [128]. Consequently, when the job queue buffer reaches its maximum capacity, new jobs arriving at the buffer are dropped until space becomes available. However, the FIFO scheduling policy may experience significant performance issues in scenarios where available resources are insufficient for the next job in the queue [129]. In such cases, the affected job, along with any subsequent ones in the buffer in FIFO order, must wait. This situation can lead to an instance where the buffer has jobs that could be serviced, but the servers remain idle until all earlier-arriving jobs have been assigned the required resources. This phenomenon, known as HOL blocking, occurs when the first job in the queue cannot be served due to insufficient resources. As a result, all subsequent jobs, even those with minimal resources demands, are blocked [124][130]. The implications of HOL blocking in such a system are multifaceted: It reduces the throughput by interrupting job processing, increases latency as jobs are uniformly delayed regardless of their size and source, leads to resource under-utilization due to resource idling while waiting for the HOL job to be served, and increases the

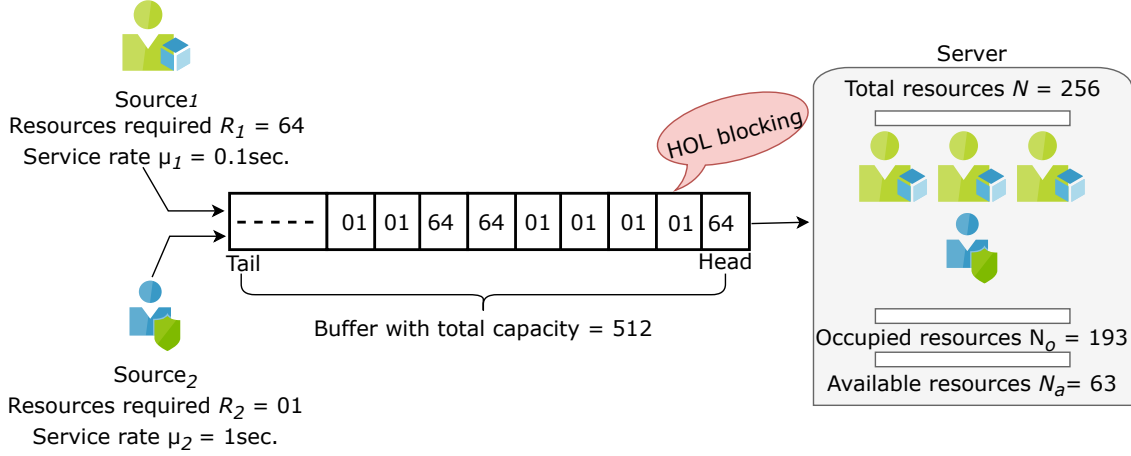


Fig. 5.1: *FIFO Queueing with the HOL Blocking*

probability of job drops when the buffer reaches its maximum capacity in systems with finite buffer capacities. This is especially problematic when the job's required resources vary significantly, as jobs requiring fewer resources to complete their service could be processed during the waiting period [131].

The HOL blocking in FIFO is affected by several factors, these include the time required to finish a job (service time) and the resource requirement of big jobs at the head of the queue. As shown in Figure 5.1, the FIFO scheduling model may experience the HOL blocking where jobs from Source₁ (requiring 64 cores for 10 seconds) and Source₂ (requiring a single core for a unit of service time) are processed in the order they arrive in the buffer. The HOL blocking is indicated, showing the delay and under-utilization when big jobs from Source₁ at the HOL require more resources than are currently available. This situation forces the subsequent small jobs to wait for their turn, resulting in several problems. Specifically, it increases the waiting time for all jobs in the queue and reduces overall system throughput.

5.4.2 Hard Priority Scheduler

This scheduling policy diverges from FIFO by focusing on high-priority jobs in the buffer, which, in our case, are the 'Big Jobs'. Big jobs refers to the jobs which requires highest amount of required resources and service time, i.e., jobs from Source₁. Whenever resources are free, the server prioritizes these high-priority jobs to receive the required resources, reducing their waiting time [132]. This means that if even one 'Big Job' is in the queue awaiting resources, 'Small Jobs' will not receive service. As with the FIFO policy, a notable downside of this approach is the potential

occurrence of HOL blocking particularly for lower-priority jobs, which may experience extended delays. Moreover, these delays tend to be longer in scenarios where the arrival probability or service time of high-priority jobs is significantly greater than that of low-priority jobs. The blocking in this approach is influenced by the job arrival frequency and the service time of the big jobs. It can lead to prolonged delays for small jobs if big jobs are frequent and consume most of the resources. Figure 5.2, depicts the Priority queueing model where Source₁'s jobs (big jobs) are given precedence over Source₂'s small jobs. HOL blocking occurs when the queue contains at least one big job awaiting sufficient resources, holding up the processing of small jobs.

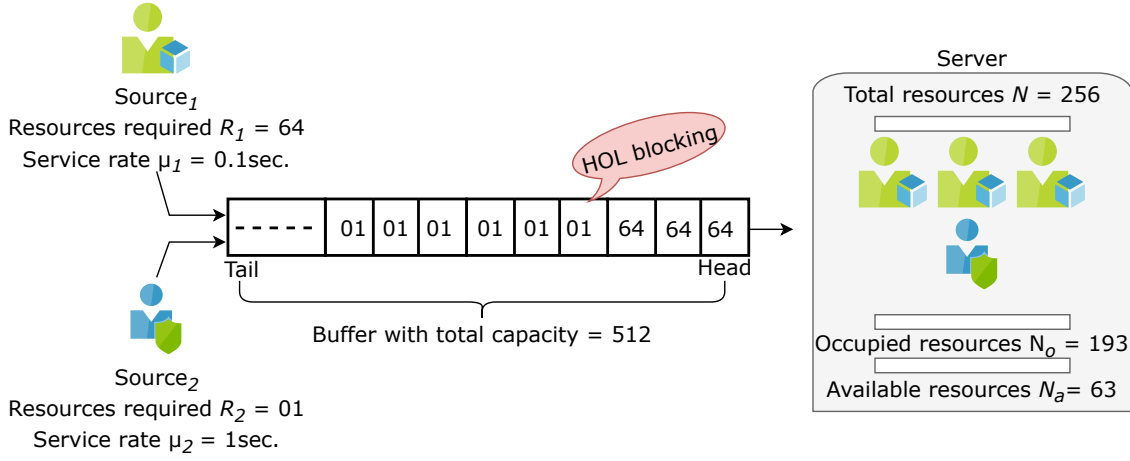


Fig. 5.2: *Priority Queueing with the HOL Blocking*

5.5 Simulation Model and Implementation

We used OMNet++ [126] simulator (Version 6.0.2) to simulate the queueing model described in the previous sections. The parameters for the simulation are:

- λ : job arrival rate measured in jobs per second;
- μ_s : service rate for small jobs, measured in jobs per second;
- μ_b : service rate for big jobs, measured in jobs per second;
- p : probability that a job belongs to the class of small jobs;
- r_b : number of resources required by the class of big jobs;

- r_s : number of resources required by the class of small jobs;
- R : total resource capacity of the system;
- B : available positions in the buffer excluding the jobs in service. Notice that in the buffer, each job occupies only one position regardless of its origin or resource requirements. In other words, both big and small jobs each take up a single position in the buffer.
- Scheduling policy: the discipline used by the system to decide which is the next job to be served (FIFO or Priority in our case).

It is notable that the big jobs in our simulation model require 64 resources for 10 seconds, while the small jobs require 1 resource for a unit of time to complete their service, respectively. Here, a question arises: Why is the total resource capacity a multiple of the resource requirement of big jobs? The answer is that in simulations, any resource arrangement of resource capacity or resource requirement can work. However, we choose this particular setting because if the maximum resource capacity is not a multiple of the resource required by big jobs (64 in our case), the remaining resources will be consistently occupied by small jobs (1 in this setting). For example, if the total resource capacity of the system is 300 and the big jobs require 64 resources to complete their service, then 256 resources will typically be occupied by big jobs, leaving 44 resources that will likely be occupied by small jobs. This creates an imbalance in resource allocation.

Once the simulation model parameters are specified, the simulator automatically executes the model until the predetermined simulation time limit is reached. In this work, to account for variability, each simulation consists of 30 independent experiments. Then, confidence intervals can be built based on the experiment's outcomes. The nature of simulation in the current study is in a single-threaded fashion, where the events are processed one at a time in a sequential manner. Currently, the model can process approximately 6,000,000 events within 360,000 seconds when executed on a MacBook Air A2337 with an M1 chip and 8GB of RAM. Hereafter, we use the parameterization of the model shown in Table 5.1.

5.6 Simulation Results and Insights

In this section, we present the impact of the arrival rate and the mixture of small and big jobs on the probability of job dropping. The analysis methodically quantifies the average frequency of dropped jobs, the average number of jobs in service, and the

Table 5.1: *System Parameters*

Parameters	Value
Buffer Capacity B	512
Service Rate μ_b [j/s]	0.1
Service Rate μ_s [j/s]	1.0
Available Servers R	256
Size of big jobs r_b	64
Size of small jobs r_s	01
Repetitions	30
Sim-Time Limit	5000000s

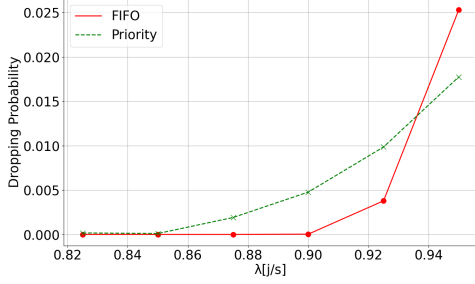
average number of jobs waiting in the buffer. Notice that the following proposition holds:

Proposition 1. *The dropping probability is the same for small and big jobs.*

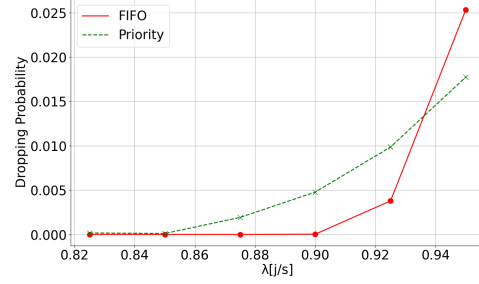
Proof. Since jobs arrive according to an independent Poisson process and they belong to the small or big class according to an independent Bernoulli experiment with p probability of success. By the splitting property of the Poisson process, the streams of big and small jobs are independent Poisson processes. Now, whenever the system saturates, the newly arriving jobs are dropped at the buffer. Recall that by the PASTA property which is a central property in queueing theory. It states that the probability an arriving job observes the system in a particular state is equal to the long-run time-average probability of the system being in that state [27][133]. For example, if a job arrives and looks at the current state of the system, what it observes will be representative of how the system behaves on average over a long period. This property holds true for any type of job arriving in the system, whether it is a large job, a small job, or belongs to different job classes. Therefore, both big and small jobs see, at their arrival time, the same statistics on the system state and share the same job dropping probability.

Figure 5.3 shows the job-dropping probability for the two considered disciplines. The data is segmented based on the allocation of arrival probabilities with a split of 40%-60% ($p = 0.6$) and 10%-90% ($p = 0.9$) for big and small jobs, respectively,

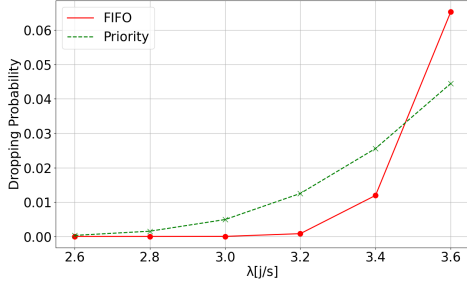
thereby offering a comparative overview of the queueing performance under varied workload distributions. This reflects their infrequent yet significant presence in real-world cloud environments.



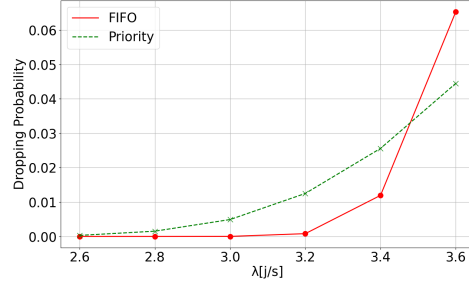
(a) *Big Job dropping probability for $p = 0.6$*



(b) *Small Job dropping probability for $p = 0.6$*



(c) *Big Job dropping probability for $p = 0.9$*



(d) *Small Job dropping probability for $p = 0.9$*

Fig. 5.3: *Dropping probability of Big and Small jobs as a function of λ*

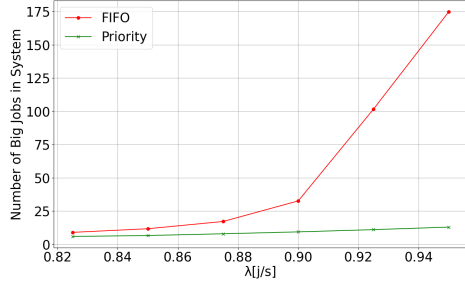
First of all, note that, as stated by Proposition 1, the dropping probability for big and small jobs is the same, and we consider this step as a sanity check for the simulator. The data presented here reveal that assigning priority to jobs with higher resource and service demands has a positive impact on the dropping probability of jobs with lesser resource and service demands in moderate and heavy load, while FIFO turns out to be the best choice in low load.

Insight 1. *Our first observation is that we waste system resources when the scheduling discipline forces the system to serve a big job after a sequence of small ones. Homogeneous sequences of small (or big) jobs do not cause a waste of resources. Therefore, ideally, a discipline should limit the changes of the phases of services between small and big jobs to improve resource utilization.*

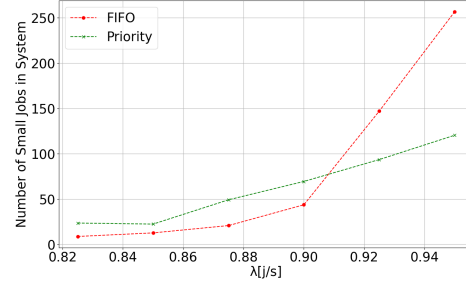
Insight 2. *The comparison between FIFO and Priority is quite interesting, revealing that there is no discipline that is better than the other in all scenarios. In low load conditions, FIFO scheduling behaves better than Priority (in terms of dropping*

probability) because the switching between small job service sequences to big ones is less frequent. Indeed, under Priority, a big job entering at the tail of the queue will cause HOL blocking if there are not enough resources, while FIFO would continue to serve the small jobs already in the buffer. On the other hand, Priority serves the big jobs one after the other, regardless of their arrival order. Therefore, since in heavy loads it is likely to find more than one big job in the buffer, this reduces the frequency of switching between big and small jobs service phases (and, hence, also the opposite as evident from Figures 5.4a 5.4c).

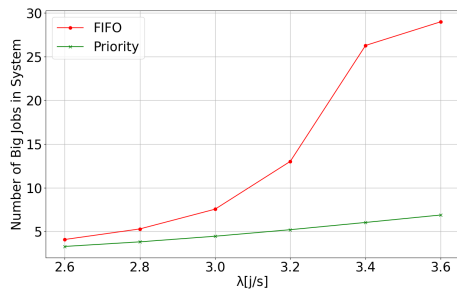
The average tenancy in the system for the Big and Small jobs is shown in Figure 5.4, respectively. Recall that the average occupancy is strictly connected to the expected response time. While the x-axis uses raw (λ , in jobs/s), this metric is directly proportional to ρ under fixed μ , as it scales linearly with λ when μ remains constant. For example, in systems with $\mu = 10$ jobs/s (a common assumption in queueing analysis), an arrival rate of $\lambda = 8$ jobs/s corresponds to $\rho = 0.8$. This implicit link to utilization states queueing dynamics using arrival rate or occupancy rather than ρ explicitly. By retaining λ as the x-axis, we emphasize how absolute workload impacts dropping behaviour in systems with heterogeneous job sizes (e.g., "big" vs. "small" jobs), while still enabling readers to infer utilization through the known relationship $\rho = \lambda/\mu$.



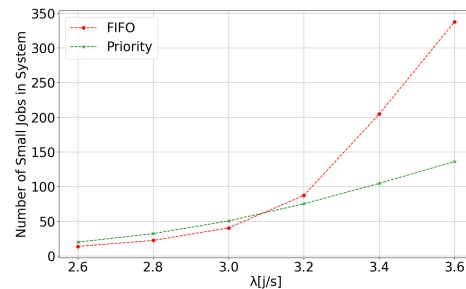
(a) Expected no. of Big Jobs in the system for $p = 0.6$



(b) Expected no. of Small Jobs in the system for $p = 0.6$



(c) Expected no. of Big Jobs in the system for $p = 0.9$



(d) Expected no. of Small Jobs in the system for $p = 0.9$

Fig. 5.4: Expected occupancy in the system as a function of λ

Insight 3. Whereas, giving priority to big jobs does not improve the probability of dropping with respect to small jobs, big jobs have a lower occupancy under the Priority policy with respect to FIFO. This is well supported by intuition. The situation is more fascinating for what concerns the small jobs. In fact, we notice that under moderate load conditions ($\lambda = 3.6$ j/s) the average occupancy of small jobs is lower under Priority scheduling than the FIFO; however, also the dropping probability is lower for priority, and this means that the throughput of small jobs is higher. In conclusion, we can state that for $\lambda = 3.6$ j/s both big and small jobs have a smaller expected response time under Priority than FIFO. This seems quite counter-intuitive but can be explained by the better resource utilization obtained by the Priority policy under moderate and heavy loads.

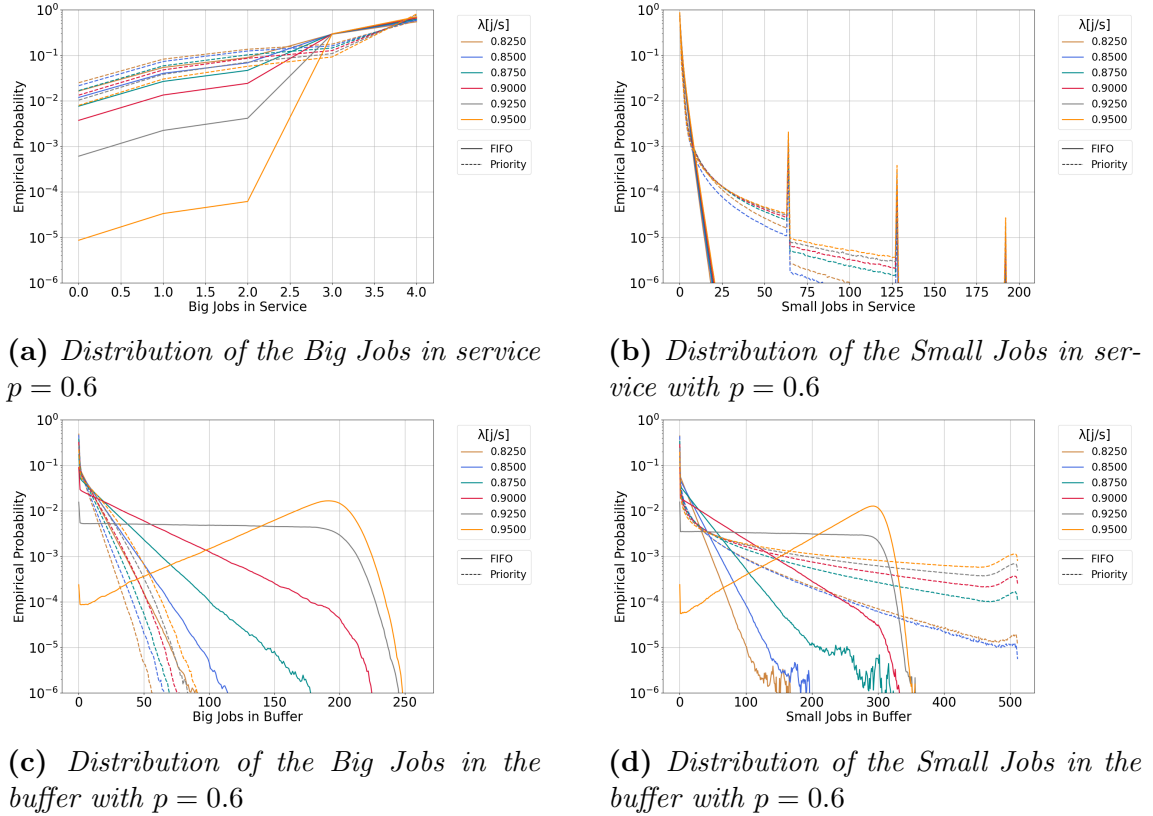


Fig. 5.5: Jobs in service and in the buffer with $p = 0.6$.

Figures 5.5a and 5.5b show the distribution of jobs in service with $p = 0.6$. Under the FIFO policy, the probability distribution is smoother and increases steadily as the arrival rate increases. Here, the Priority policy heavily favors big jobs, which can lead to resource starvation for small jobs, especially at high arrival rates. As the system's arrival rate increases, the difference in the treatment of big and small jobs becomes more drastic under the Priority policy. The notable transient spikes at a

regular interval of small jobs in service in figure 5.5b depict the difference between the resources required by big jobs and small jobs, which is 64 vs. 1 in our case.

Figures 5.5c and 5.5d show the buffer state as a function of λ . Here, it can be seen that Priority scheduling heavily favors big jobs, leading to a significant number of small jobs accumulating in the buffer, especially as the system's arrival rate increases. However, for big jobs, Priority scheduling minimizes their waiting time in the buffer, which is ideal if big jobs are less delay-tolerant.

We acknowledge the instability of the low empirical probabilities ($< 10^{-5}$) in Figure 5.5d. These values correspond to rare events (e.g., prolonged starvation of small jobs under extreme loads) and are inherently noisy due to finite simulation time limit. While extending simulation time could stabilize these estimates, the observed trend aligns with theoretical expectations for priority queueing systems with resource asymmetry. We retained this range to highlight the qualitative behavior of extreme tail probabilities, which are critical for systems requiring strict QoS guarantees.

Furthermore, figures 5.6a and 5.6b show the probability of the average number of jobs in service when the arrival probability of big and small jobs is in the ratio of 10%-90%.

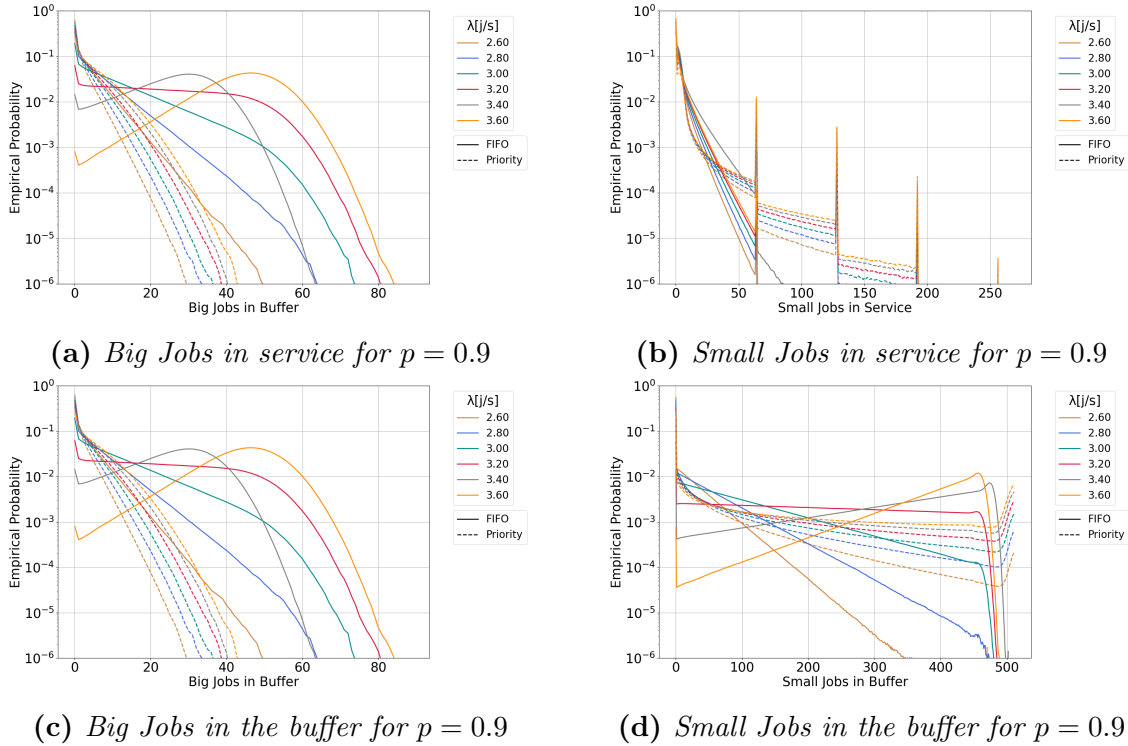


Fig. 5.6: Jobs in service and in the buffer with $p = 0.9$.

The insight that can be drawn from these plots is that the Priority model strongly favors big jobs and big jobs do not often wait in service, while small jobs, despite

their high arrival rate and low resource requirements, are at a disadvantage in the Priority model, as their service is postponed whenever big jobs are present. The spike immediately before the discontinuity reflects a transient bottleneck phenomenon: As λ approaches system capacity, small jobs briefly accumulate in service before being abruptly displaced by large jobs (which require 64 cores).

This behavior aligns with priority queueing models, where varying resource demands create abrupt transitions in service allocation near saturation. Similar observations can be drawn from Figures 5.6c and 5.6d considering the scenario with $p = 0.1$.

5.7 Conclusion

In this chapter, we propose a simulation study of the MJQM with finite-capacity of resources and buffer. We employed this setting with two different scheduling disciplines, FIFO and Priority. Since in open models, it is not trivial to determine the stability region of the queueing system with infinite capacity, employing finite queues can help to overcome this problem. However, in this case, the analysis of the impact of the buffer size on the percentage of dropped jobs becomes a crucial aspect to address.

The simulator handles an arbitrary number of classes and resources, but this investigation was limited to the simplest scenario involving two job classes: Small Jobs and Big Jobs. The big jobs require 64 resources for 10 seconds each, whereas the small jobs need just 1 resource for 1 second to complete the service. Additionally, our study was constrained by the consideration of only a single type of resource referred to as cores, with every incoming job requiring the same resource type. In the finite buffer, each job either, it is big or small occupies a single position.

Our simulations show some counterintuitive effects of priority scheduling (i.e., serving Big Jobs with priority with respect to the Small ones) that reveal that the introduction of this feature allows for a lower dropping probability of both Big and Small Jobs in moderate and heavy load conditions. Even for the expected response time, giving priority to Big Jobs allows for a smaller expected response time for Small ones thanks to the better utilization of the resources produced by the serialization of the service of big jobs.

Chapter 6

Scaling Multi-Server Job Systems: A Dynamic Approach with 29 Job Sources

6.1 Introduction

In recent times, emerging computing models have transformed the dynamics of cloud computing into highly complex and heterogeneous environments. These advancements have intensified the diversity of job sources, job types, and their resource demands. For example, an AI-driven application requires dynamic resource allocation to handle fluctuating computational processing, whereas an IOT device generates latency-sensitive jobs that strain traditional scheduling disciplines. Improving the reliability of such systems can be achieved by carefully analyzing and modeling key elements such as job scheduling, waiting, service times for these services, and the likelihood of hardware and software failures [118].

6.2 Motivation

The explosion of data and the need for ultra-fast processing have highlighted the importance of a dynamic job and resource scheduling policy. In job scheduling literature, various scheduling policies are proposed, yet service providers in complex cloud environments still find it challenging to select the appropriate algorithm for their applications due to the dynamic nature of cloud resources and their allocation. We extended our work in Chapter 5 to simulate dynamic job sources using our OMNeT++-based simulator. The motivation behind doing so is to explore further

insights into resource allocation dynamics. We extended the simulation model developed in our previous work and tailored it to handle dynamic job classes [100]. This model is designed to accurately represent the dynamic nature of cloud resources and their allocation.

6.3 Dynamic Queueing Model Description

The varying requirements of jobs from dynamic sources result in relatively low utilization [127]. In order to investigate the impact of dynamic job sources with variable arrival and service rates in finite-capacity queueing systems, we employed the MJQM-F with the following characteristics: (i) we have N identical resources; (ii) jobs arrive at the system belonging to one of 29 classes, the minimum resource required by a job is 1 while the maximum resources required by a source is 2998 in this dataset of Google Cell A in appendix [134]; (iii) jobs arrive according to a time-homogeneous independent Poisson process, where the highest arrival probability of a job is 0.93963536 while the lowest arrival probability is 0.000029246; (v) each job has a service time that is exponentially distributed with class-dependent rates, where the minimum service time in this dataset is 0.049277s while the maximum service time required by any job is 385.76513s. The job scheduler decides to assign the resources to one or more jobs in the buffer based on the scheduling discipline defined in the configuration file. The next section describes the scheduling policies that we used to evaluate the performance of this dynamic queueing model.

6.4 Scheduling Policies Evaluated

The above-stated queueing model is simulated with FIFO. For detailed description, algorithm, and mathematical notation of the FIFO scheduling discipline, readers are referred to 4.2.4 and 5.4.

6.5 Simulation Model Evaluation

To assess the performance of the proposed model, we applied the same performance evaluation metrics as declared in Section 4.2.6. Table 6.1 provides details of the parameters used for this simulation.

Table 6.1: *Simulation Parameters*

Parameter Name	Value
Simulation Time	17000s
Number of Simulation Runs	30 Times
Buffer Capacity	2000 units, each job take one unit of space in buffer
Resource Capacity	4096 units
Scheduling Policy	FIFO
λ Range	450, 550, 650 and 750

6.6 Results

Since the proposed simulator can handle dynamic job sources, we simulated this configuration for different arrival rates (λ) ranging from 450 to 750 jobs/s and computed different metrics such as dropping probability, number of jobs waiting in buffer, number of active jobs, response time, and resource utilization for the FIFO-based scheduling policy. The results show that the dropping probability increased from 0.020% to 27% as λ increased from 450 to 750 jobs/s as shown in Figure 6.1. Looking at the arrival probability, required resources, and service time of the Google traces from the appendix, we see that Source 1 generates 94% jobs due to its higher arrival probability; however, it only requires a single resource out of 4096 (maximum capacity of the system); thus, its high volume dominates the arrival stream of the system. Jobs from Source 1 are processed quickly (due to short service time, i.e., 0.049277s), but their sheer number fills the finite capacity of the buffer rapidly, especially at high arrival rates (e.g., 750 jobs/s). This indirectly causes dropping for other sources when the buffer saturates because the buffer considers all the jobs uniformly without any classification. Similarly, the jobs from source 8 have an extremely long service time (e.g., 385.76513s) so even rare arrivals block the buffer and resources for extended periods, causing cascading drops for subsequent jobs. Therefore, at lower arrival rates, i.e., $\lambda = 450\text{--}550$ jobs/s, the drops are minimal ≤ 0.05 as the system handles the load efficiently. However, the dropping probability

increases sharply (e.g, 0.05–0.15) due to buffer contention from Source 1’s high-volume jobs. The dropping probability plot reflects a system trapped between high-volume trivial jobs and rare but disruptive heavy jobs (as shown in Figure 6.1).

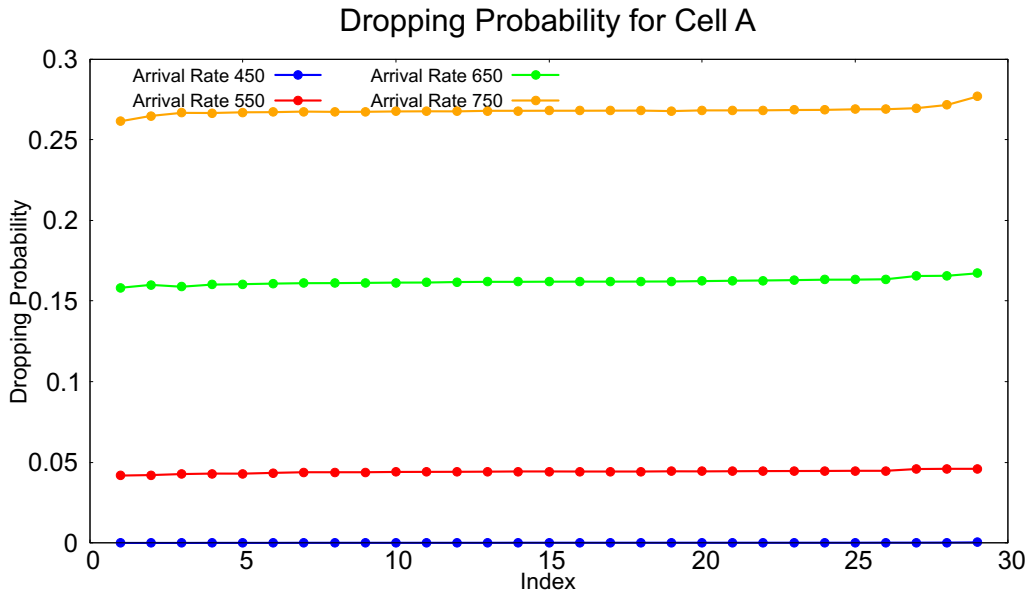


Fig. 6.1: *Dropping Probability for different arrival rates in CellA.*

The figures 6.2 and 6.3 refer to jobs waiting in the buffer and active jobs in process. Here, it is evident that the buffer occupancy increases with arrival rate, but plateaus at 2000 for the highest loads. For $\lambda = 450\text{--}650$ jobs/s, jobs enter the buffer but are processed swiftly to prevent buffer overflow. At $\lambda = 750$ jobs/s, the buffer is often full, so the average occupancy stabilizes near the maximum capacity of the buffer. The high variability in buffer occupancy at high arrival rates suggests frequent transitions between full and near-full states of the buffer.

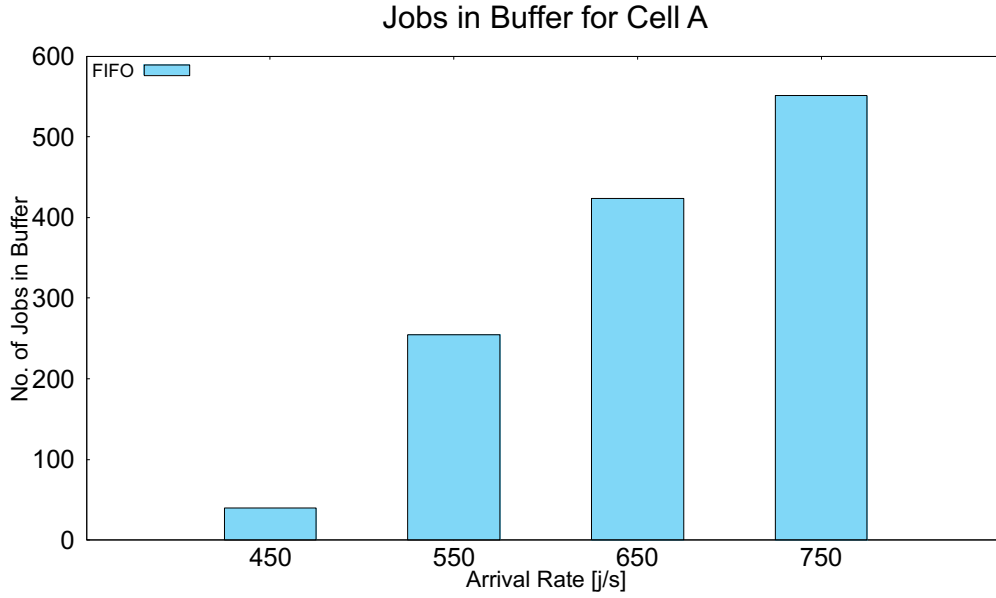


Fig. 6.2: *Jobs in Buffer for different arrival rates in CellA.*

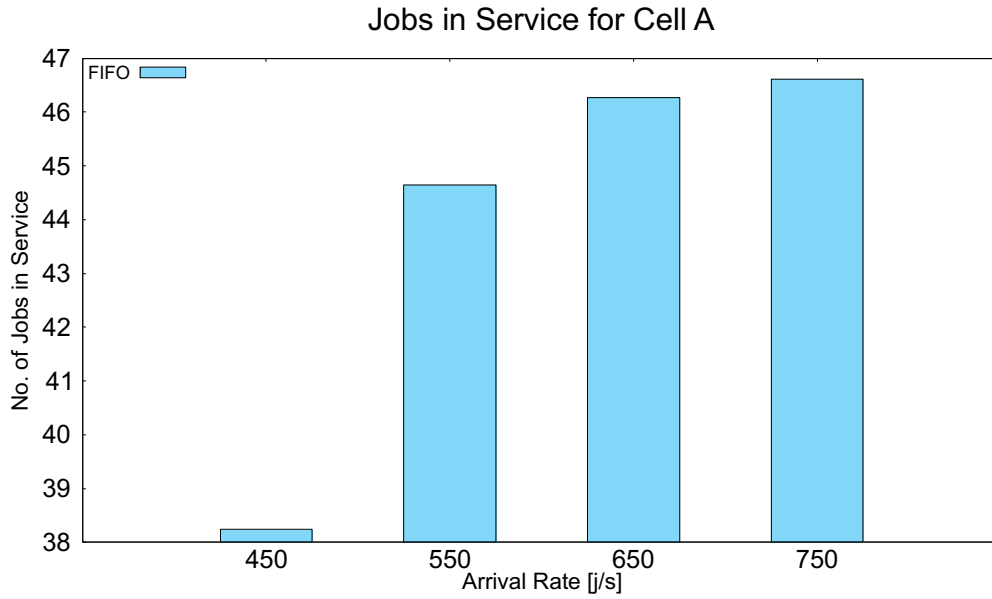


Fig. 6.3: *Jobs in Service for different arrival rates in CellA.*

Similarly, for the number of jobs in service (as shown in Figure 6.3), it initially increases with the arrival rate but plateaus due to resource constraints. The system can process jobs in parallel using up to 4096 resources. If jobs require ≤ 1 resource each, up to 4096 jobs can be in service. However, if jobs require more resources, the plateau occurs earlier, which is our case. At high arrival rates, resource contention

likely causes the number in service to stabilize, as jobs wait for resources even if buffer space is available.

Figure 6.4 shows the completion time for each job source with different arrival rates in ascending order with the log scale on y-axis. Due to finite capacity of the buffer and the resources available in the system, most of the jobs spend an identical amount of time waiting for required resources; however, the service time in the dataset from the Appendix varies from 0.049s to 385.765s due to which we witness the variation in response time.

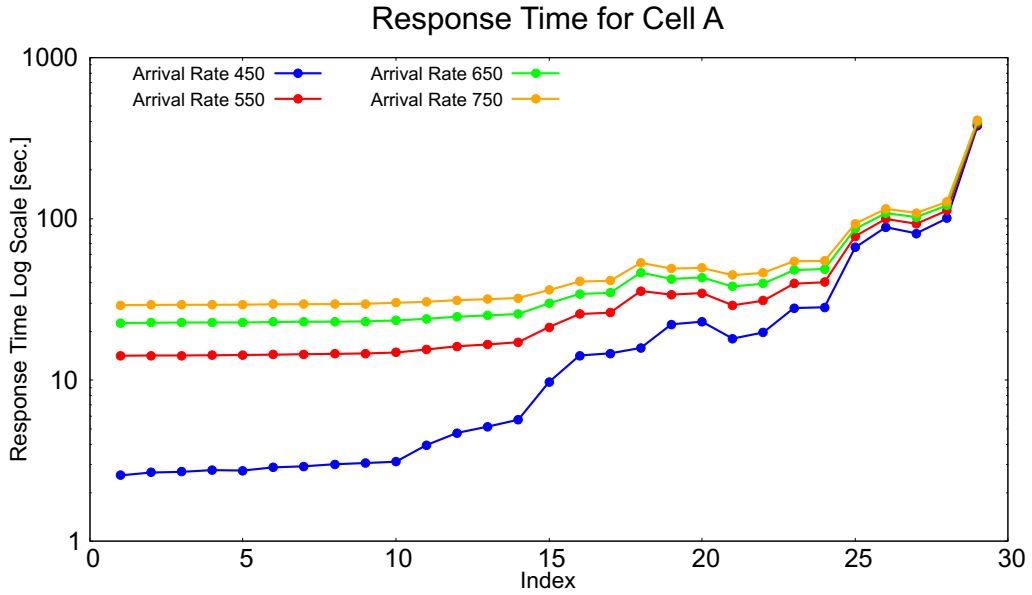


Fig. 6.4: *Response Time for different arrival rates in CellA with Log Scale.*

Figure 6.5 represents the resource utilization of dynamic sources and it can be observed that as the arrival rate increases, more jobs enter the system, reducing the idle time between resource allocations. FIFO processes jobs sequentially, so a steady influx of jobs ensures that resources are utilized more consistently.

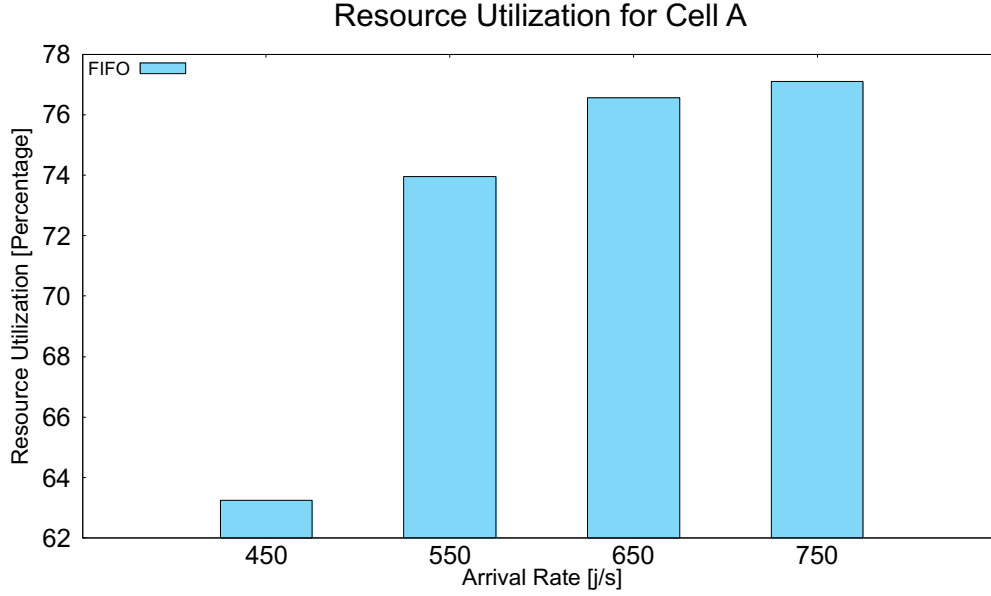


Fig. 6.5: *Resource Utilization for different arrival rates in CellA.*

6.7 Conclusion

Our proposed simulator elaborated in Chapter 4 can handle an arbitrary number of classes and resources. In this chapter, we simulated the MJQM-F with dynamic sources with FIFO scheduling. The total number of job sources in our simulation model was 29. Our FIFO-scheduling based simulations demonstrate distinct performance thresholds under varying arrival rates (450–750 jobs/s). At lower rates (≤ 550 jobs/s), the system operates stably with minimal job drop, low response times, and moderate resource utilization. However, with increasing arrival rate, the system approaches saturation: buffer occupancy nears its maximum capacity, dropping probability spikes, and response times escalate exponentially, reflecting severe queueing delays. Resource utilization plateaus at 75%, indicating inefficiencies from contention and fragmentation, while jobs in service stabilize at 46, constrained by resource availability. These trends highlight a transition from buffer-limited to resource-limited performance at higher loads. The optimal operational range lies between $\lambda = 550$ –650 jobs/s, balancing throughput and reliability. Beyond $\lambda = 650$ jobs/s is the hot area and the trade-offs between performance degradation and resource allocation must be carefully managed.

Chapter 7

Performance Evaluation of Multi-Server Job System with Preemption and Restart

7.1 Introduction

The increasing demand for data processing and storage has led to the rapid growth of large-scale cloud data centers, termed hyper-scale data centers [135]. Owned and operated by companies such as Google, Alibaba, Amazon, Samsung, and Apple. They serve a variety of purposes that include data analytics, data storage, data processing, AI, automation, and other computing operations on big data. These hyper-scale data centers are expanding in number and size worldwide. These data centers consumes more energy than most countries, with their annual electricity consumption is surpassed only by the four largest economies [67][136][137]. This extraordinary energy footprint not only impacts the environment but also reinforces the importance of optimizing energy efficiency and resource management within these facilities. Therefore, the need for optimization directly relates to the system's performance metrics, such as the length of the jobs waiting for service and resource utilization as a whole. To address inefficiencies and improve performance metrics, resource providers have increasingly turned to alternative job scheduling and resource allocation techniques instead of relying solely on the FIFO discipline [136].

7.2 Background and Motivation

The increasing demand for data storage and job processing has led to the introduction of complex data centers that are termed Hyper Scale Data Centers. In job scheduling literature, researchers have proposed various scheduling policies, yet service providers in cloud environments still find it challenging to select the appropriate algorithm for their applications due to the dynamic nature of cloud resources and their allocation. This difficulty arises from the heterogeneity of resource types (memory, bandwidth, storage, I/O and so on), inter-dependencies, uncertainty, and dispersion of assets in the cloud environment.

The authors of [138] proposed a preemptive priority-based computationally efficient method for analyzing multi-job queues. Their method reduced state complexity, and they validated its accuracy through numerical experiments, especially for Poisson arrival system. Similarly, the work by [139] provided an exact solution for multi-server queues with preemptive priorities, delivering precise performance metrics but with high computational demands and limited generalization to real-world scenarios. The study in [140] introduced the Urgent Job First with Backfilling algorithm for preemptive scheduling in heterogeneous systems, focusing on high-priority computing tasks but facing challenges in a broader and complex implementation. The work in [141] presented a multi-server queueing model with randomized preemptive priorities, offering performance insights but suffering from limited empirical validation and application complexity.

While these studies offer valuable insights into preemptive scheduling in different contexts, they generally focus on theoretical models and specific use cases, leaving their applicability to dynamic large-scale cloud computing environments limited. This research aims to bridge this gap by evaluating the impact of preemptive and non-preemptive scheduling disciplines in the cloud computing environment, particularly focusing on key metrics like dropping probability, resource utilization, and energy wastage introduced due to preemption. By doing so, this research extends the applicability of these concepts to more complex and practical scenarios. The sections ahead refer to the discussion on the underlying scheduling model, its parameters for analyzing the impact of preemption and restart on job scheduling, and the corresponding performance metrics. The contributions of the proposed work are summarized as follows:

1. **Development of a comprehensive simulation model for Preemptive and Non-Preemptive FIFO scheduling**

We extended the simulation model developed in our earlier work and tailored

it to handle more than two job classes and diverse requirement types such as bandwidth, cores and storage [126][100][37]. This model is designed to accurately represent the dynamic nature of cloud resources and their allocation. To promote further research and collaboration, we have made the codebase open source and available on our Github public repository in Section 7.7.

2. In-depth analysis of queueing policies

Our study provides a detailed evaluation of different scheduling disciplines. These include FIFO and Greedy FIFO scheduling policies. In Non-Preemptive FIFO the jobs arriving in the buffer are allocated available resources in the order of their arrival. In Greedy FIFO, the acquired resources are freed by first preempting smaller jobs from the active job list and gradually progressing to larger jobs until the threshold of the required resource for the head of buffer job is met. The simulation results revealed a significant improvement in the performance metrics with a minimal increase in energy wastage.

7.3 Model Description

The reliability of the cloud can be improved by modeling several aspects of this system. These include scheduling, service time, waiting time, hardware, and software failures. The waiting time for jobs in cloud computing environments is a important metric that directly impacts overall efficiency and user experience [142]. We extended the MJQM-F from Section 5.3 to investigate the impact of job scheduling on waiting time and resource utilization with the following characteristics: (i) We have N identical servers; (ii) Jobs arrive at the system from one of 29 sources, derived from Google Borg’s cluster traces [134][143][144]; (iii) These jobs have diverse resource requirements, ranging from as low as 1 server to over 73.19% of the total resource capacity, making them highly resource-intensive; (iv) Jobs arrive according to a time-homogeneous independent, Poisson process; (v) Each job has a service time that follows exponential distribution with a source-dependent rate. The job scheduler decides which job to pick up from the available jobs pool in the buffer depending on the underlying scheduling policy. The detailed characteristics of the dataset for CellA, including resource requirements, arrival probabilities, and service times, are provided in Appendix A. Notably, Cell A exhibits significant skewness, with 94% of jobs requiring ≤ 1 resource unit. This reflects real-world heterogeneity, where small jobs dominate in frequency and large jobs dominate in resource consumption. The skewness directly impacts scheduling dynamics: large jobs drive

prioritization disciplines, while small jobs intensively compete for leftover resources. This distribution underscores the need for adaptive scheduling to balance efficiency and fairness.

In the following subsection, we describe the FIFO and Greedy FIFO scheduling, which we compare using our simulator, with the real data traces from the Cella dataset mentioned earlier. We simulated this work with different β values, a weighting factor that determines whether preemption should occur. The simulation analyzed dropping probability, waiting time, energy wastage due to preemption, and resource utilization for deeper insights.

7.4 Scheduling Policies Evaluated

The above-stated queueing model with numerous input sources is simulated with the FIFO and Greedy FIFO scheduling policies. The following sections discuss the analysis of all policies with respect to the parameters of our simulation model. It is worth noticing that, in the real data center environment, the arrival probability and required service time of the workload are not known in advance, and this is assumed from the statistics of similar experiments carried out earlier.

7.4.1 FIFO

FIFO is the simplest scheduling discipline based on a single rule of processing the first job that arrives and letting it complete its service without preemption. For more details and a mathematical overview of the problem formulation of the FIFO scheduling policy, the readers are encouraged to read Sections 4.2.4 and 5.4.

7.4.2 Greedy FIFO

The second policy we implemented in our model extends FIFO by introducing preemption. This scheduling policy is designed to address resource under-utilization in workload management.

The Greedy FIFO diverges from the FIFO in that this policy preempts active jobs (e.g., jobs currently being processed) when specific conditions are met. It aims to enhance resource utilization by preempting one or more active job(s), releasing its acquired resources, and re-adding the preempted jobs to the tail of the buffer. The released resources are summed with already free resources to serve the job at the head of the buffer. Here, the service progress of preempted jobs is reset to the initial

status, and thus they restart their service whenever the resources are granted for the next time, which means this preemption is based on restarting the service.

The process of identifying an active job or subset of active jobs for preemption is initiated whenever there are jobs waiting in the buffer for sufficient resources, the resource utilization is not 100%, and preemption will increase the resource utilization. The increase in utilization refers to the difference between utilization $\rho^d(t)$ at time t and post preemption utilization ρ_{post}^d , the utilization after preemption. This is mathematically presented in Equations 7.1 to 7.4. These preemption decision-making conditions are visually summarized in Figure 7.1.

The decision to preempt either a single or a subset of active jobs is made on the basis of cost-benefit analysis-comparing the cost of killing a single job or the entire subset with the cost of leaving resources idle as shown in the Equation 7.4. Equation 7.5 computes the killing cost of a single job which is a product of the resources acquired by a job $r_{x,i}$ and the time spent in service $Age_{x,i}$. In case more than one job is the potential candidate for preemption, this cost is calculated for all the candidate jobs. The rationale for selecting killing cost as the primary criterion for preemption decision as it has the ability to balance short-term efficiency and long-term fairness by accounting for both the magnitude and duration of resource consumption. Killing cost inherently penalizes jobs that monopolize resources for longer periods, even if their resource requirements are low. By factoring in the age in service, killing cost discourages frequent preemptions of long-running jobs, which can incur significant overhead. However, we are looking forward to exploring a possible weighted criterion that considers both resource requirements and termination costs could offer a more balanced approach to scheduling.

Similarly, the cost of leaving the currently leftover resources un-assigned (idling cost) is a product of Expected($\bar{s}_{x,i}$) service time of the HOL job from buffer and the leftover resources at time t and it is presented in Equation 7.6. The expected service time which is a product of arrival probability of job $p_{x,i}$ and required service time $\bar{s}_{x,i}$ of the HOL job in the buffer as shown in Equation 7.7. The preempted jobs are moved back into the buffer if space allows. If the buffer space is insufficient to accommodate all preempted jobs, they are added in the increasing order of their arrival time. Similarly, if only a single position in the buffer is available, only the oldest one from the preempted subset of jobs is kept in the worst case and the rest are discarded and counted as number of jobs dropped from their respective origin in the simulation statistics. The rationale for this selection strategy of retaining the oldest job in the system, rather than the newly arrived job, is to improve QoS. Dropping a job that has already spent a significant amount of time in the system,

as opposed to a newly arrived job, would result in energy wastage. This preemption helps improve resource utilization; however, preempting a job from service results in energy wastage. This energy waste for Greedy FIFO is computed as elaborated in Equation 7.8. It refers to the total energy wasted during the simulation whenever an active job is preempted and moved back into the buffer or discarded in the worst-case scenario, as discussed above during the simulation.

Algorithm 8 presents the rationale behind the selection of a subset of active jobs for preemption. It ensuring that the required resources for the HOL are sufficient. This process begins by checking if there is a job at the head of the buffer and determining the resources it requires. If no such job exists, the algorithm terminates. Otherwise, it calculates the initial resource usage and available free resources, then sorts the active jobs in ascending order of their resource requirements and descending order of their service start times. Iterating through this sorted list, the algorithm accumulates free resources and computes the total killing cost. The loop terminates when the accumulated resources meet or exceed the resource requirements of the HOL job.

A cost-benefit analysis is performed and the algorithm proceeds with preemption by calling the implementation procedure in Algorithm 9; otherwise, preemption is skipped. This ensures efficient resource utilization while minimizing energy wastage. Besides this, Algorithm 9 refers to the logic for implemented the preemption decision made. This includes removal of active jobs from service into buffer. Resources are released as jobs are removed from the active jobs list. The holding Queue is sorted by job arrival times, and the job in the head of the buffer is processed by starting its service. The algorithm attempts to reinsert jobs from the holding queue back into the buffer if space allows; otherwise, jobs are dropped, and drop counts are updated. So, during the simulation, the model continuously assigns the resources to the jobs at the head of the buffer, and whenever the resources are underutilized and the above-stated conditions are met, an active job preemption is carried out. To elaborate more, let us express this policy with mathematical notation which is an extension of mathematical model discussed in Section 4.2.6.

- The system's buffer length $\neq 0$.

$$B \neq 0 \quad \text{Buffer has no jobs} \quad (7.1)$$

- There are spare resources:

$$\begin{cases} \mathbf{R}(t) = \mathbf{R} - \sum_{(x,i) \in \mathcal{S}(t)} \mathbf{r}_i, \\ \mathbf{R}(t) > \mathbf{0} \quad (\text{component-wise}). \end{cases} \quad (7.2)$$

- The HOL job in the buffer can be served with the resources released by any active job when combined with the already available resources.
- Preemption will result in higher resource utilization than current utilization.

$$\rho_{\text{post}}^d > \rho^d(t) \quad (7.3)$$

- The killing cost of the job $<$ the idling cost:

$$\sum_{(x,i) \in S} \text{Killing Cost}_{x,i} < \text{Idling Cost} \quad (7.4)$$

where S is a subset of active jobs (could be a single job).

The killing and idling cost are calculated as follows:

$$\text{Killing Cost}_{x,i} = r_{x,i} \times \text{Age}_{x,i} \quad (7.5)$$

$$\text{Idling Cost} = \text{Expected}(\bar{s}_{x,i}) \times R(t) \quad (7.6)$$

$$\text{Expected}(\bar{s}_{x,i}) = \sum_{i=1}^m p_{x,i} \bar{s}_{x,i} \quad (7.7)$$

Where:

- $r_{x,i}$: Number of resources held by job x from origin i .
- $\text{Age}_{x,i}$: The age of the job x from origin i (that is, how long it has been running).
- $\bar{s}_{x,i}$: Mean service time of the job x from the source i (time taken to process each job).
- $p_{x,i}$: Arrival probability of job x from class i .
- $R(t)$: Number of idle resources when considering job i .

- m : Total number of job classes in the model.

- Preemption of active job(s) results in rework and other computational waste which is computed as follows:

$$\text{Energy Waste} = \frac{\sum_{i=1}^m \text{Killing Cost}_{x,i}}{\text{Sim-Time Limit}} \quad (7.8)$$

Here:

- *Sim-Time Limit*: Simulation time limit.

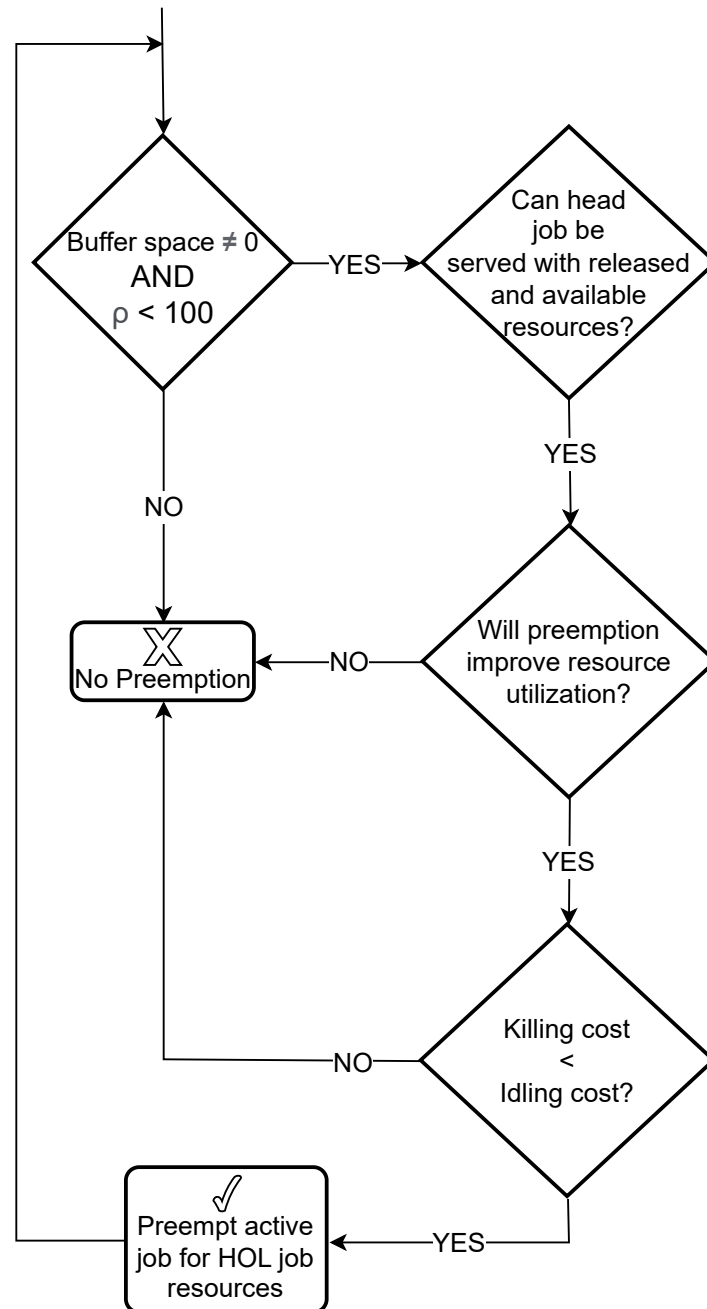


Fig. 7.1: Flowchart illustrating the conditions for preempting an active job based on cost-benefit analysis.

Algorithm 8 Greedy FIFO Preemption Algorithm to Identify the Subset of Jobs to Preempt

```

1: Function identify_fifo_greedy_preemption.job()
2:    $\alpha \leftarrow 1.0, \beta \leftarrow 0.0$ 
3:   headJob  $\leftarrow$  buffer.checkNextJob(resourceCapacity)
4:   if headJob is null
5:     return
6:   requiredResources  $\leftarrow$  headJob.requiredResources
7:   initialResourceUsage  $\leftarrow$  sumOfResourcesUsedByActiveJobs()
8:   totalFreeResources  $\leftarrow$  resourceCapacity
9:   sort activeJobs by:
10:    ascending requiredResources, descending serviceStartTime
11:   for each job in sortedActiveJobs
12:     totalFreeResources  $+=$  job.requiredResources
13:     totalKillingCost  $+=$  job.requiredResources  $\times$  (simTime -
        job.lastServiceStartTime)
14:     if totalFreeResources  $\geq$  requiredResources
15:       break
16:   if totalKillingCost  $<$  idlingCost
17:     call implementation_fifo_greedy_preemption(preemptCandidates, resource-
        Capacity - initialResourceUsage, headJob.serviceTime)
18:   else
19:     skip preemption

```

7.5 Simulation Model and Implementation

We used the OMNet++ [126] simulator (Version 6.0.2) to simulate the queueing model and scheduling policies described in the previous sections. Once the parameters are specified, the simulator automatically executes the scheduling model until the predetermined simulation time limit is reached. To account for variability, each simulation consists of 30 independent experiments for Cella. To explore the behavior of preemption and killing cost, we simulated this setting for different β values. This is the cost coefficient that determines the relative importance of idling cost compared to the killing cost.

Algorithm 9 Greedy FIFO Preemption Implementation Algorithm

```

1: Function implementation_fifo_greedy_preemption(preemptJobs, initialFreeRe-
   resources, headJobServiceTime)
2:   idlingCost  $\leftarrow \beta \times \text{initialFreeResources} \times \text{headJobServiceTime}$ 
3:   totalKillingCostForThisPreemption  $\leftarrow 0$ 
4:   for each job in preemptJobs
5:     totalKillingCostForThisPreemption  $+= \alpha \times \text{job.requiredResources} \times (\text{sim-}$ 
      Time  $- \text{job.lastServiceStartTime})$ 
6:   if totalKillingCostForThisPreemption < idlingCost
7:     increment totalPreemptions
8:     totalKillingCost  $+= \text{totalKillingCostForThisPreemption}$ 
9:     holdingQueue  $\leftarrow$  empty list
10:    for each job in preemptJobs
11:      cancel scheduled end-service message for job
12:      remove job from activeJobs
13:      initialFreeResources  $+= \text{job.requiredResources}$ 
14:      add job to holdingQueue
15:      sort holdingQueue by firstArrivalTime (oldest first)
16:      headJob  $\leftarrow$  buffer.popNextMessage(resourceCapacity)
17:      if headJob is not null
18:        call startNextJob(headJob)
19:      for each job in holdingQueue
20:        if buffer.isFull()
21:          drop job (increment drop count, delete job)
22:        else
23:          insert job into buffer
24:    else
25:      skip preemption (killing cost is too high)

```

The nature of simulation in the current study is in a single-threaded fashion, where the events are processed one at a time in a sequential manner. Hereafter, we use the parameterization of the model shown in Table 7.1. Detailed information on arrival probability, service time, and required resources for each class of jobs for Cell A is available in the appendix (See Table 9.1).

Table 7.1: *System Parameters*

Parameters	CellA
Buffer Capacity B	20480
Arrival Rate λ_i [j/s]	550
Available Servers R	4096
Sim-Time Limit	4000s
Job Sources	29
Repetitions	30
β Range	0 - 30.0

7.6 Simulation Results and Insights

These statistics are from the simulation of traces of CellA with FIFO and Greedy FIFO scheduling disciplines. The simulation consists of 30 repetitions, resulting in over 4 million events per iteration. The total λ of the system is 650 jobs per second. Figures 7.2 and 7.3 show the dropping probability. Figure 7.2 depicts the dropping probability of different β values. Here $\beta=0$ depicts the dropping probability of the system with zero killing of jobs or we can say that it resembles the FIFO job scheduling discipline.

This results in the highest dropping probability (0.1622), as expected, due to lack of job preemption and queue congestion. Similarly, non-zero β value refers to preemption condition where higher β value amplifies the weight of the idling cost, making resource under-utilization more expensive. This means that the algorithm will prioritize avoiding idling costs over other considerations. Since the idling cost becomes higher, the preemption condition is more likely to be satisfied.

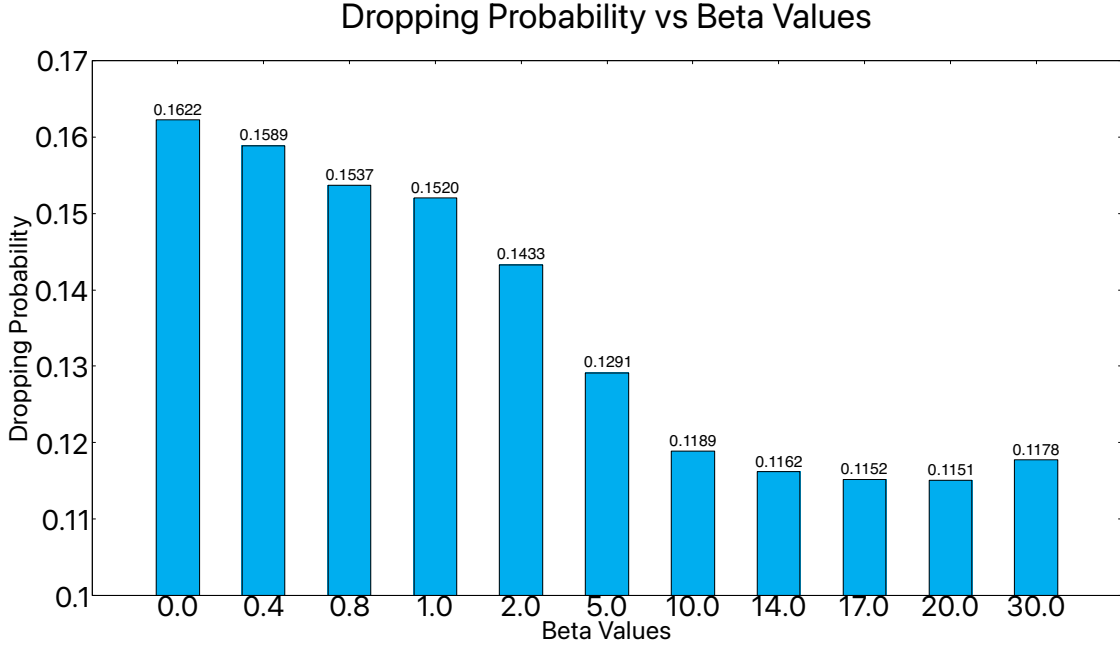


Fig. 7.2: *Dropping Probability for different Beta Values*

As a result, the algorithm will preempt active jobs more frequently to free up resources for the HOL job in the buffer and will result in higher utilization. When the β range from 0.4 to 20, we observed a monotonic decrease in dropping probability (e.g., 0.1622- 0.1151). Here, the active job preemption increasingly prioritizes resource utilization, reducing queue saturation. However, at maximum β , the model exhibits a slight increase in probability of dropping (0.1151 to 0.1178). This increase suggests that a trade-off in the model with overly aggressive preemption at very high Beta values may destabilize the system. So we can comment that $\beta = 20$ achieves the lowest dropping probability, balancing, and sufficient preemption to maintain resource utilization with minimal disruption to job processing. Overall, this model demonstrates the relationship between job preemption and dropping probability for resource utilization and highlights the need to tune β to avoid over-optimization. This aligns with real-world systems, where aggressive policies often return suboptimal results.

Figure 7.3 dropping probability of job sources with respect to β values. Here, it is worth noting that, for better visual presentation we skipped the job sources, which depicts a similar trend. Here, the different trend for source 4 and source 8 is due to their high service-time requirements from their neighboring job sources.

Figure 7.4 presents the analysis of job waiting time. This time underscore the importance of tuning β based on the specific performance metric prioritized in the

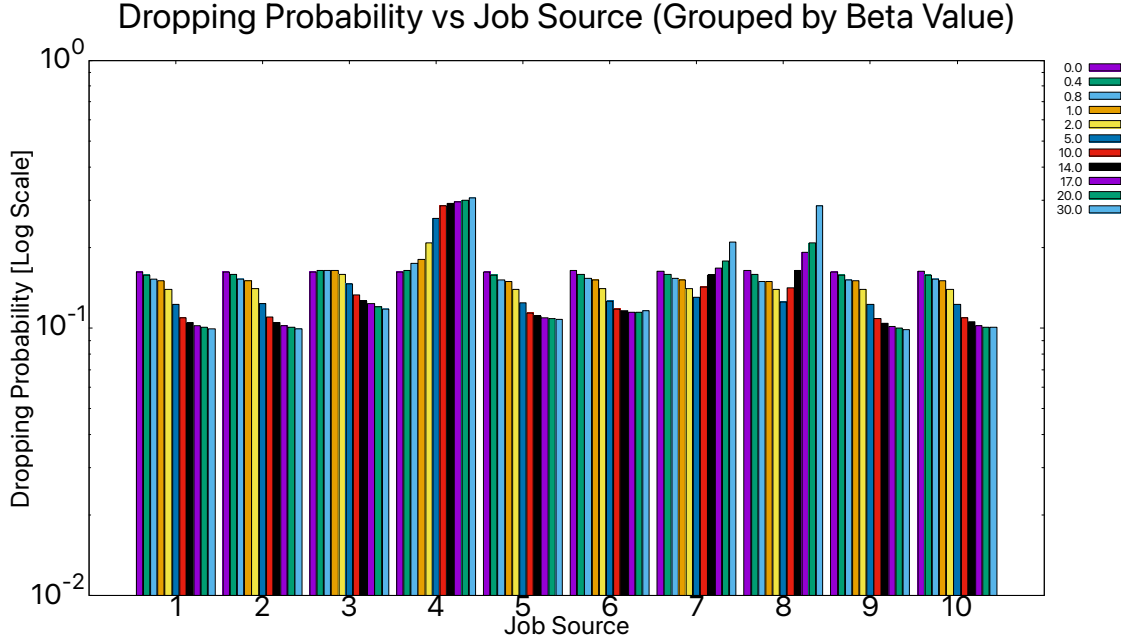


Fig. 7.3: *Dropping Probability for Each Source*

system. Similarly to the dropping probability, the waiting time is maximum for $\beta=0$ because of no preemption FIFO. Here, the waiting time is the highest, which aligns with the expectation that no preemption leads to poor resource utilization, causing jobs to remain in the buffer for longer periods. Similarly, when we tune the β between 0.5 and 5.0 the waiting time steadily decreases as Beta increases (e.g., 23.3786 to 15.4804 seconds). It happened because the preemption improves resource utilization, reducing congestion in the buffer and allowing jobs to be processed more efficiently.

However, when β is in the range between 10.0 and 30.0, the waiting time starts to increase again after $\beta = 5.0$, peaking at $\beta = 30.0$ (20.7026 seconds). This increase suggests a trade-off in the system dynamics: In the moderate range, i.e., $\beta = 0.50$, the system achieves the optimal balance between preemption and resource utilization, minimizing waiting time. Here, jobs are preempted just enough to avoid queue saturation without disrupting processing flow. Similarly, for high β (e.g., 30.0) excessive preemption destabilizes the system, leading to frequent interruptions, contention for resources, and aggressive preemption. From this fact, it can be analyzed that $\beta = 5.0$ achieves the optimal waiting time, due to sufficient preemption to reduce congestion.

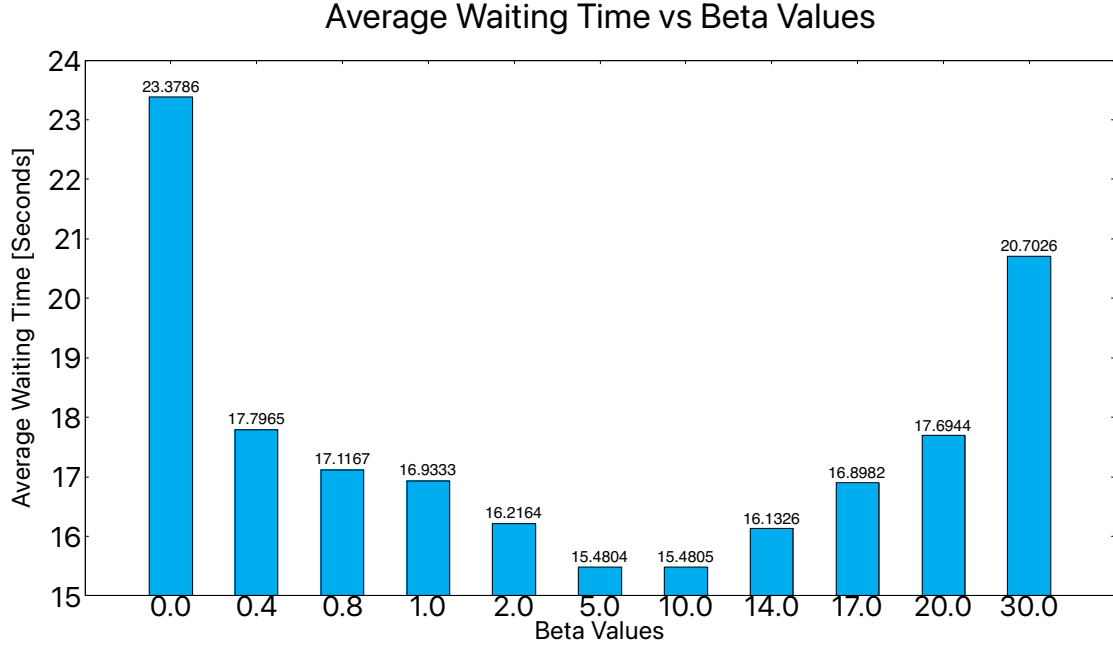


Fig. 7.4: *Waiting Time for Different Beta Values*

To avoid redundancy and improve clarity, we selected only job sources that demonstrated distinct variations in waiting time across different values of β for visualization in Figure 7.5. From this figure, the high waiting bar for source 8 is due to the high required service time, so for a high β value this source spends much time waiting in the buffer. We noticed that the optimal β for waiting time differs from the same for dropping probability. This highlights a trade-off that the lower tuning values prioritize shorter waiting times by avoiding excessive preemption. Higher β values prioritize lower dropping probabilities by aggressively preempting jobs to free resources.

In complex systems, optimal resource utilization is highly desired. Fig 7.6 depicts the resource utilization of proposed study. The trend here aligns with the expected behavior of our model: as β increases, the idling cost becomes more significant, making the algorithm more aggressive in preempting jobs to avoid resource under-utilization. At $\beta = 0$, resource utilization is 76.55% , which represents the baseline utilization without any preemption incentives. As we tune the β to higher range, resource utilization continue to improves, reaching 85.62% at $\beta = 30.0$.

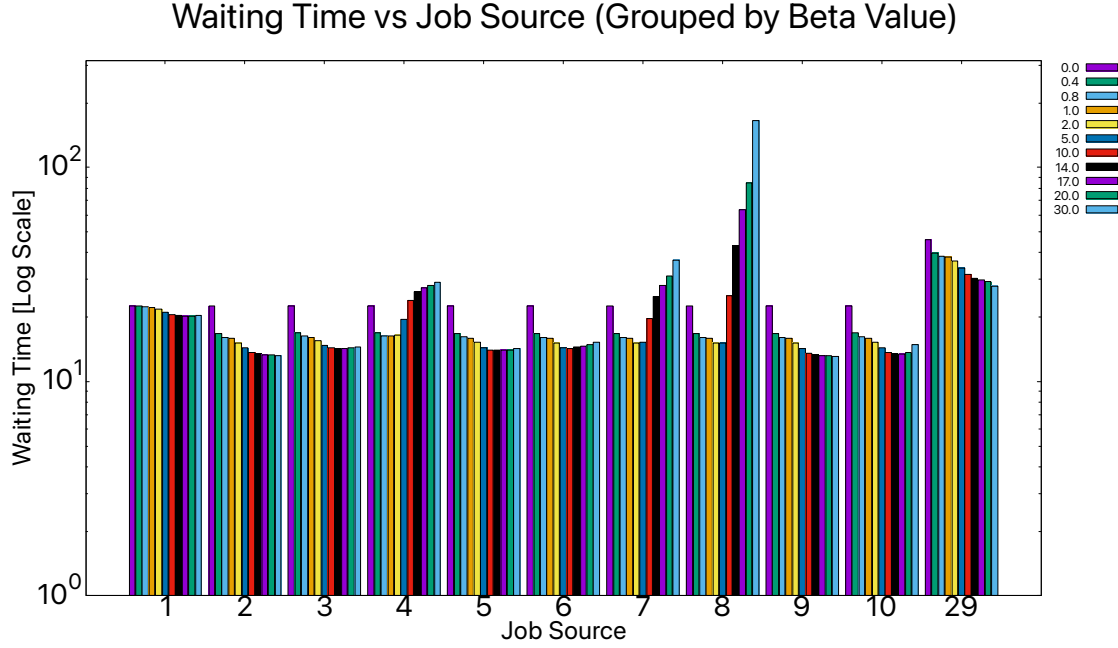


Fig. 7.5: *Waiting Time for Each Source*

However, the rate of improvement in resource utilization decreases as β increases, with gains diminishing at higher values; for instance, utilization increases by 2.22% from $\beta = 0$ to $\beta = 2.0$, but only by 1.27% from $\beta = 20.0$ to $\beta = 30.0$

Although higher β values improve resource utilization, they also introduce trade-offs that need to be carefully considered:

- **Computational Wastage:** A higher tuning range leads to more frequent active job preemptions, which increases computational wastage. These preempted jobs are added back to the buffer. Their service progress is reset to the initial stage, and they are restarted, consuming additional energy and computational resources.
- **Jobs Dropping:** As we set β to a higher value, more jobs are moved back into the buffer after preemption. If the buffer is full, some jobs will be dropped, potentially degrading the system's overall performance and QoS.
- **System Overhead:** Frequent preemption of active jobs results in computational overhead. This includes managing the overall preemption process, sorting jobs, and recalculating resource allocations and system metrics.
- **System Fairness:** Aggressive preemption at high β values may unevenly impact lengthy jobs, as they are more likely to be preempted and dropped frequently.

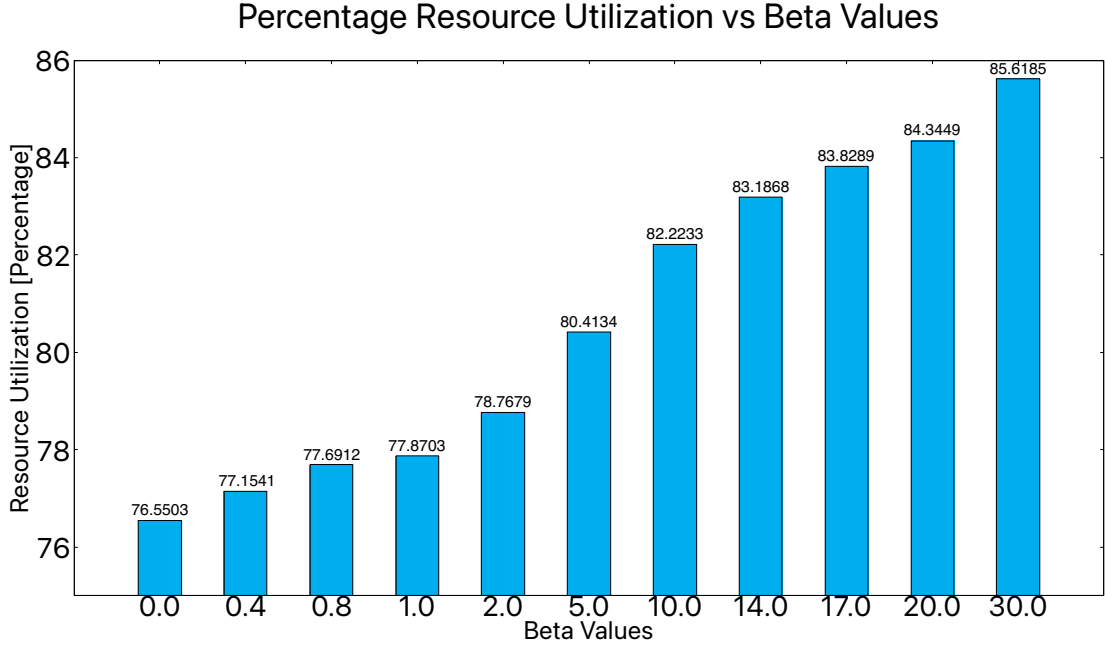


Fig. 7.6: *Resource Utilization*

Thus, the choice of β depends on the specific goals of your system. For instance, if maximizing resource utilization is the primary goal, a higher β is preferable, as it achieves near-maximum resource utilization. Similarly, for a balance between resource utilization and energy efficiency, a moderate range of β is recommended. To support the importance of selecting an optimal β value based on the requirement of the system, we compute analytical utilization (as shown in Figure 7.7) based on Eq. 7.9 which is inline with the mathematical notation declared in Section 4.2.6. Here, $P_{a,i}$ is the arrival probability of job from source i . The simulated utilization graph in Figure 7.6 reflects the actual resource-usage achieved through job preemption with varying β values, capturing the effects of rework, job drops and computational wastage. In contrast, the ideal utilization in Figure 7.7 represents the theoretical maximum resource usage calculated under the assumption that there is no rework due to preemption, providing an upper bound on the achievable utilization for each dropping probability scenario. The difference of 0.33% in $\beta = 0$ validates our model. Comparison of these graphs underscore the gap between ideal performance and real-world constraints. This gap is caused due to preemption and system limitations.

$$\rho_{\text{Ideal}} = \frac{\sum_{i=1}^m \left(\lambda_i \cdot P_{a,i} \cdot (1 - P_i) \cdot r_i \cdot \frac{1}{\mu_i} \right)}{R} \quad (7.9)$$

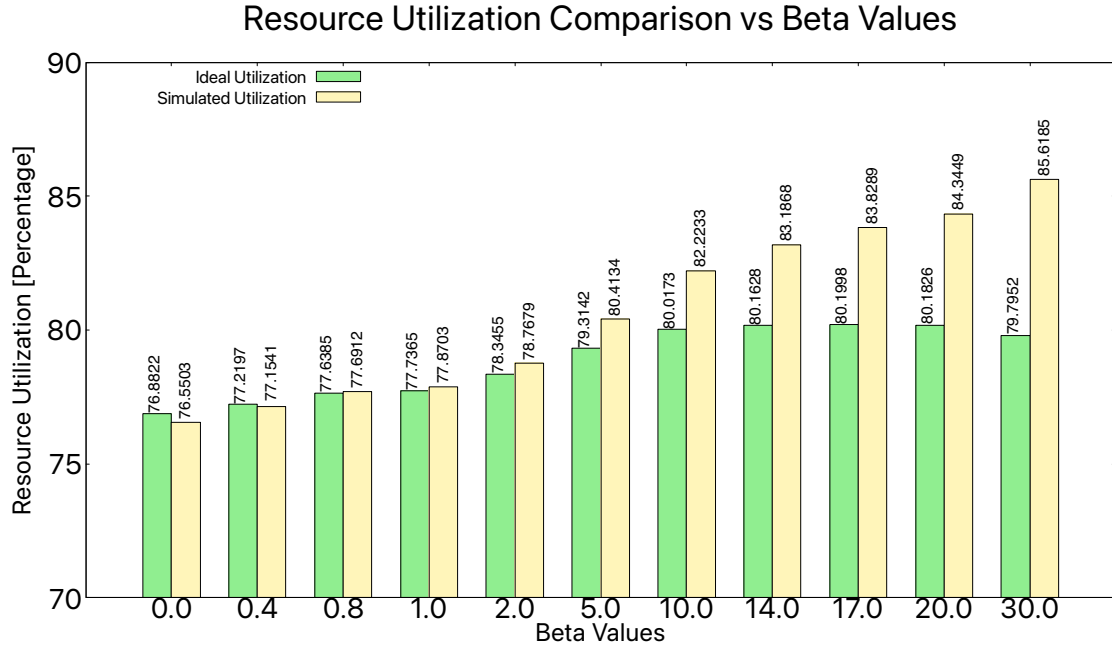
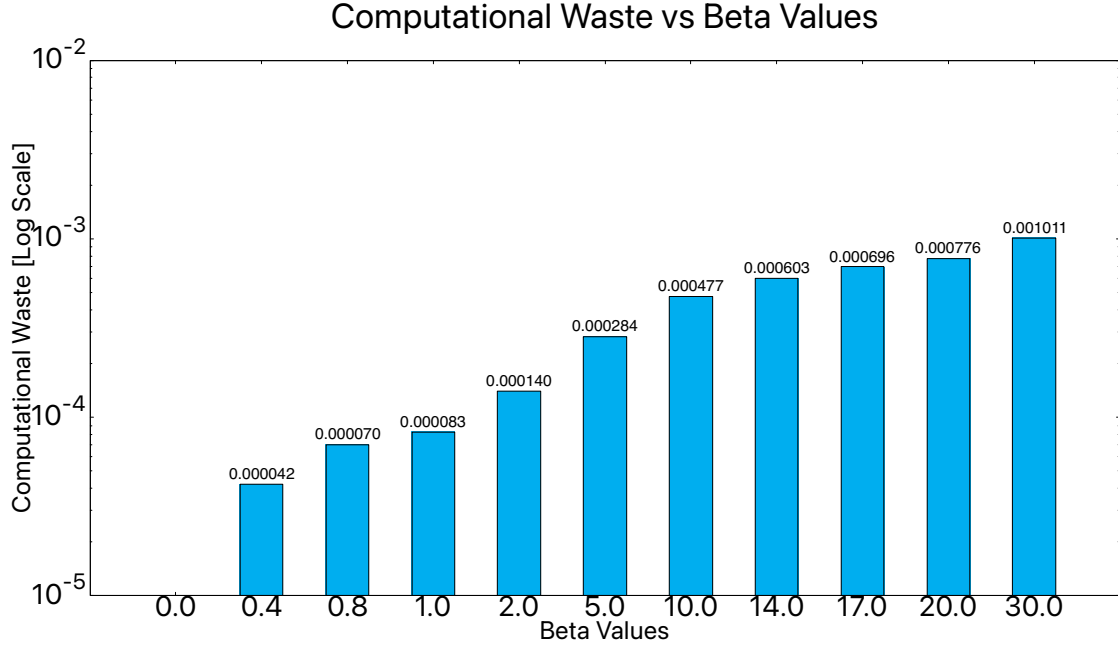


Fig. 7.7: *Resource Utilization Comparison*

As discussed earlier in this section, tuning of β introduces some trade-offs. Taking into account the high cost and other environmental factors associated with energy, we present the computational waste of this study in Figure 7.8. At $\beta = 0$, there is no computational waste that validates that there is no incentive for preemption during the simulation. Adapting β to higher values, computational waste rises linearly, reaching 0.001011 at $\beta = 30$. This indicates that a higher weighting coefficient leads to more frequent preemption of active jobs, which results in greater computational waste. A β value of 5.0 to 10.0 is recommended for most systems, balancing resource utilization and computational waste.

Fig. 7.8: *Computational Waste*

7.7 Simulator Availability

The complete source code and evaluation results for our model with FIFO and Greedy FIFO scheduling, are publicly accessible on our GitHub repository. This includes detailed files such as the codebase, network descriptions, configurations, and result files. The aim of this repository is to allow others to replicate, modify, and build upon our work. We actively welcome feedback, suggestions, and collaboration opportunities from the community to enhance and expand this tool further. For further information and to access these resources, please visit our GitHub page.¹

7.8 Future Work

The current study reveals that preemptive scheduling policies such as Greedy FIFO improve resource utilization by avoiding HOL blocking of jobs in the buffer and thus mitigating resource under-utilization. However, these benefits come at the cost of computational wastage as a result of the preemption of active jobs from service.

¹FCQ Performance Evaluation Repository on GitHub: https://github.com/muhammadwaqas01/FCQ_Performance_Evaluation

As part of future work, we plan to extend the simulation study by varying job arrival rates to gain deeper insight into the relationship between resource utilization, performance metrics, and computational wastage under these different resource types and scheduling policies. By analyzing these experiments with different arrival rates and even experimenting with CellB from [134], which offers tricking statistics with a lot of variation in the data, we aim to identify strategies to minimize energy waste while maintaining optimal resource allocation.

It is also noteworthy that the FIFO discipline, which does not preempt the job to accommodate the HOL job from the buffer, results in zero energy wastage. This characteristic of FIFO provides a baseline for comparison to evaluate the trade-offs introduced by preemptive scheduling policies. The insights obtained from this extended simulation will contribute to a more comprehensive understanding of how different scheduling policies affect energy efficiency and overall system performance in cloud computing environments.

7.9 Conclusion

In conclusion, the analysis of the Greedy FIFO preemption model reveals a clear trade-off between resource utilization, computational waste, job drop rates, and system efficiency, as the parameter β is varied. At low β values (e.g., $\beta = 0-2$), the system minimizes computational waste and job drops but achieves only modest improvements in resource utilization. Whereas with moderate β values (e.g., $\beta = 5-10$) it strike an effective balance, achieving significant gains in resource utilization (80–82%) while keeping computational waste and drop rates relatively low (0.0477% and minimal dropping probability). However, at high β values (e.g., $\beta = 20-30$), while resource utilization approaches its maximum (85%), the associated costs—such as increased computational waste, higher job drop rates, and longer waiting times, highlight the lessen returns and possible decline in system performance. These findings underscore the importance of carefully selecting the optimal β based on the specific goals and constraints of the system.

Chapter 8

Conclusion and Future Work

8.1 Introduction

This chapter presents the conclusion and future work of this thesis. This research focused on optimizing cloud computing performance by simulating MJQM-F using various scheduling policies. This work contributes to an extensive understanding of job scheduling and resource allocation challenges in diverse cloud environments due to the introduction of AI-driven workloads. This provides information on how different scheduling strategies impact performance metrics such as resource utilization, drop probability, waiting time, response time, and energy waste.

8.2 Summary of Important Findings

The thesis demonstrated several key findings through the development and simulation of various scheduling policies within a cloud computing environments:

- 1. Simulation Framework and Implementation:**

A robust simulation model was developed using the OMNeT++ framework, which accurately represents the dynamics of cloud resources and their allocation. This model, which incorporates both FIFO and priority-based scheduling, was crucial for understanding the impact of different queueing disciplines on cloud system performance. The modular and flexible architecture of the simulator provided a valuable tool for analyzing real-world cloud computing scenarios.

- 2. Performance of MJQM-F:**

By simulating the MJQM-F, we found that FIFO scheduling is simple to implement, but it is also prone to HOL blocking. This leads to resource under-utilization, especially when the required resources of jobs vary significantly. On the other hand, the priority scheduling policy showed that assigning priority to jobs with higher resource requirements and longer service times requirement could improve overall system performance, positively impacting both big and small jobs.

3. Preemptive Scheduling and Energy Wastage:

The introduction of preemptive scheduling policies, such as Greedy FIFO, showed a significant improvement in resource utilization by reducing HOL blocking. However, these benefits came with the trade-off of energy wastage due to the preemption of active jobs. By preempting active jobs to free up resources, the system incurs energy waste because the service progress of these jobs resets to zero, leading to a potential increase in the probability of re-work.

4. Trade-offs and Performance Metrics:

A critical trade-off between resource utilization and energy wastage in cloud environment was identified. While preemptive scheduling improves resource utilization, it also increases energy consumption. In contrast, FIFO scheduling results in zero energy wastage due to the absence of job preemption, providing a baseline for comparison with preemptive policies. Thus, the choice of tuning β is solely dependent on the dynamics of the model under consideration.

8.3 Future Work

Building on the findings of this thesis, several avenues for future work are proposed, these include:

1. Exploring Different Arrival Rates and Dataset:

Future research will focus on extending the simulation study by varying job arrival rates to gain deeper insight into the relationship between resource utilization, performance metrics, and energy wastage, under different scheduling policies. We plan to expand this work by simulating scheduling policies using the CellB dataset from Google's Borg cluster, which is more tricky and offers significant variation in the data [134]. Analyzing these dynamics across diverse arrival rates will help identify strategies to minimize energy waste while maintaining optimal resource allocation.

2. Evaluating Advanced Scheduling Policies:

Further exploration of advanced scheduling policies, including adaptive and hybrid approaches, can provide more insights into how cloud environments can dynamically adjust resource allocation based on real-time workload conditions. This could involve incorporating ML techniques to predict job characteristics and make more informed scheduling decisions.

3. Energy-Aware Strategies:

Investigating strategies to mitigate energy wastage in preemptive scheduling policies is very important due to the challenges faced by the environment. Future work can explore methods to reduce the energy cost associated with job preemption, potentially through optimizing the decision-making process of when and which jobs to preempt, considering factors such as job age, service progress, and resource requirements.

8.4 Conclusion

This thesis provided a comprehensive analysis of scheduling policies in finite-capacity multi-server job queueing systems within cloud computing environments. A key contribution of this work is the presentation of an original, flexible, and modular OMNeT++ simulation framework for MJQM. The proposed framework aims to capture the unique dynamics of resource allocation, job scheduling, and performance metrics in heterogeneous cloud environments. This library facilitates in-depth exploration of various scheduling policies, making it a valuable tool for researchers and practitioners in cloud resource management.

Moreover, this simulator, along with all associated data, has been made freely and openly available on a public GitHub repository. The repository is made available with the aim of allowing the research community to use, replicate, modify, and extend the simulator. Researchers can identify issues, provide suggestions, and even collaborate to further enhance the, for ongoing and future studies in cloud computing.

The insights acknowledge that while preemptive scheduling policies such as Greedy FIFO can significantly improve resource utilization and reduce the HOL blocking, they introduce the challenge of energy wastage. FIFO scheduling, on the other hand, eliminates energy waste, but experiences lower resource utilization and higher job drop probabilities under heavy load conditions.

The research contributes to a deeper understanding of the trade-offs inherent in cloud resource management and scheduling policies. It highlights the significance of considering both energy efficiency and performance in the design of scheduling strategies for cloud computing systems. The findings gained from this work serve as a foundation for future research aimed at developing more adaptive, efficient, and sustainable cloud resource allocation mechanisms.

Chapter 9

CellA Dataset from Google Borg

9.1 CellA Dataset

The CellA dataset used in this study includes columns for required resources, arrival probability, and service time. Table 9.1 presents the complete dataset and Figures 9.1- 9.3 show the graph of the arrival probability, required resources, and service time of the dataset.

Sr No	Required Resources	Arrival Probability	Service Time
1	1	0.939635366	2.140987347
2	2	0.021369679	3.106702186
3	3	0.003654944	12.10234378
4	4	0.002032627	64.05097202
5	5	0.002850667	11.61818487
6	6	0.000602993	17.03883006
7	7	0.000252035	79.64336897
8	8	9.29004E-05	385.7651318
9	9	0.004762007	0.54709024
10	10	0.00068127	86.2710552
11	11	9.63412E-05	2.588492402
12	14	0.000974594	0.139611088
13	18	0.0091851	0.445713286
14	20	0.000293325	98.60385028
15	24	6.02132E-05	20.57429126
16	27	0.005891436	0.310996597
17	28	0.001886395	0.09660738
18	30	5.24715E-05	19.77444131
19	38	0.001105343	0.218118958
20	40	0.00263562	1.37944965
21	42	0.000939327	0.162844548
22	50	0.000425794	25.46456836
23	60	5.76327E-05	25.93499384
24	80	3.35474E-05	15.23780456
25	98	0.000127308	0.376461847
26	100	4.90308E-05	7.159567431
27	120	3.09668E-05	0.049277756
28	498	0.000191822	0.480738477
29	2998	2.92464E-05	0.325075486

Table 9.1: *CellA Dataset from Google Borg*

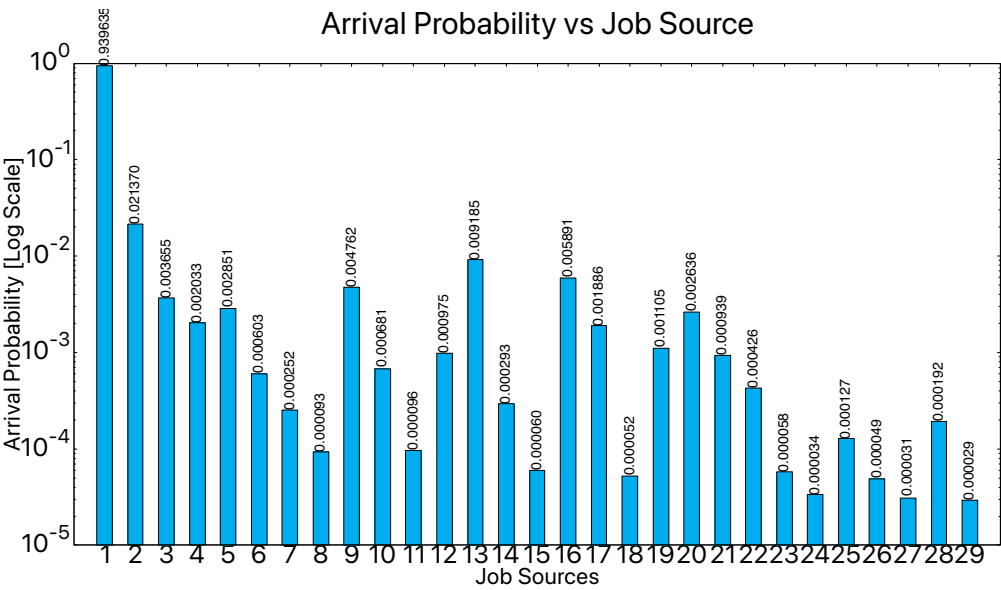


Fig. 9.1: Arrival Probability of Jobs in Cella.

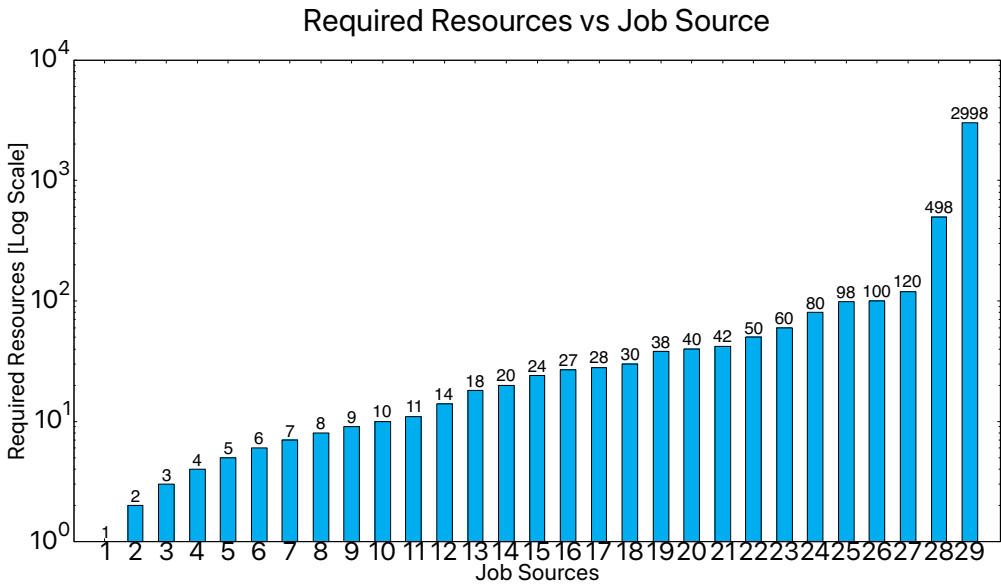


Fig. 9.2: Required Resources of Jobs in Cella.

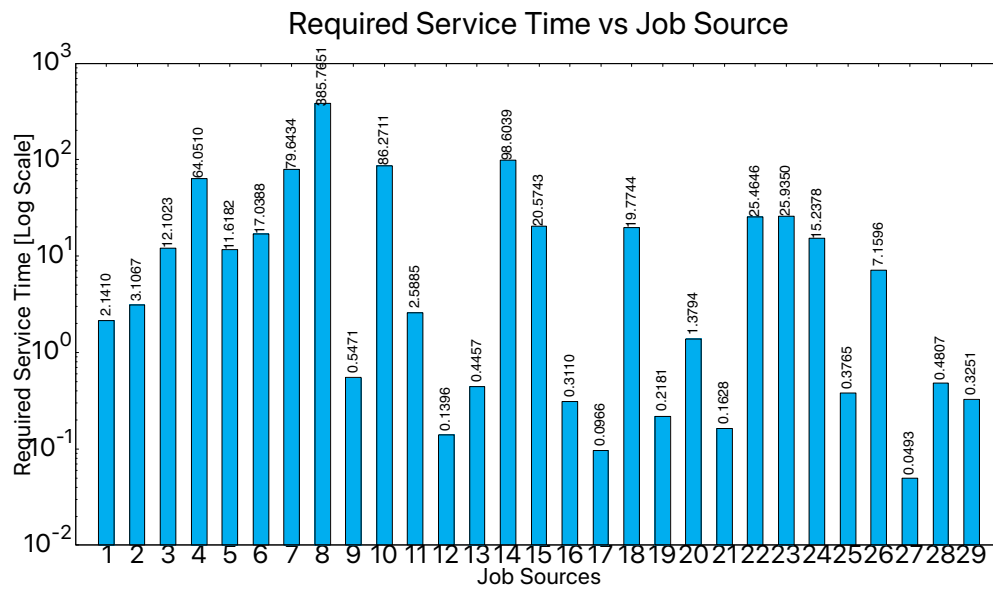


Fig. 9.3: *Required Service Time of Jobs in Cella.*

Bibliography

- [1] A. Raja Santhi and P. Muthuswamy. Industry 5.0 or industry 4.0s? introduction to industry 4.0 and a peek into the prospective industry 5.0 technologies. *International Journal on Interactive Design and Manufacturing*, 17, 2023.
- [2] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. *NIST Special Publication*, 800-145, 2011.
- [3] Kamal Dahbur, Bassil Mohammad, and Ahmad Bisher Tarakji. A Survey of Risks, Threats and Vulnerabilities in Cloud Computing. In *Proceedings of the 2011 International Conference on Intelligent Semantic Web-Services and Applications*, 2011.
- [4] Mohit Kumar, S.C. Sharma, Anubhav Goel, and S.P. Singh. A Comprehensive Survey for Scheduling Techniques in Cloud Computing. *Journal of Network and Computer Applications*, 143, 2019.
- [5] Guangyao Zhou. *Optimization Scheduling Technologies for Multi-dimensional Resources and Distributed Pipeline Parallel Training in Cloud Computing*. Doctoral dissertation, University of Electronic Science and Technology of China, 2023.
- [6] Tianwei Zhang. *Detection and Mitigation of Security Threats in Cloud Computing*. Doctoral dissertation, Princeton University, 2017.
- [7] Marco Guazzone. *Power and Performance Management in Cloud Computing Systems*. Doctoral dissertation, University of Torino, 2011.
- [8] U. Narayan Bhat. *An Introduction to Queueing Theory: Modeling and Analysis in Applications*. Birkhäuser, 2008.
- [9] Mor Harchol-Balter. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013.

- [10] János Sztrik. *Basic Queueing Theory: Foundations of System Performance Modeling*. GlobeEdit, 2016.
- [11] C. Newell. *Applications of Queueing Theory*. Springer Science & Business Media, 2013.
- [12] H. T. Papadopoulos, Cathal Heavey, and Jimmie Browne. *Queueing Theory in Manufacturing Systems Analysis and Design*. Springer Science & Business Media, 1993.
- [13] Agner Krarup Erlang. The Theory of Probabilities and Telephone Conversations. *Nyt tidsskrift for matematik*, 20, 1909.
- [14] Agner Krarup Erlang. Solution of Some Problems in the Theory of Probabilities of Significance in Automatic Telephone Exchanges. *Post Office Electrical Engineer's Journal*, 10, 1917.
- [15] B. Filipowicz and J. Kwiecień. Queueing Systems and Networks. Models and Applications. *Bulletin of the Polish Academy of Sciences, Technical Sciences*, 56, 2008.
- [16] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris. *Fundamentals of Queueing Theory*. John Wiley & Sons, Inc., 2008.
- [17] Leonard Kleinrock. *Queueing Systems, Volume I: Theory*. John Wiley, 1975.
- [18] Mor Harchol-Balter. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013.
- [19] Siyang Ding. Analysis and Application of Computer Queueing Theory. In *Proceedings of the 2023 International Conference on Image, Algorithms and Artificial Intelligence*, 2023.
- [20] Daniel P. Heyman. Book Review: Queueing Systems, Volume 1: Theory by Leonard Kleinrock. *Networks*, 1976.
- [21] Nguyen Thi Mai and Duong Manh Cuong. The Application of Queueing Theory in the Parking Lot: A Literature Review. In *Proceedings of the International Conference on Emerging Challenges: Business Transformation and Circular Economy*, 2021.

- [22] Jack Li, Calton Pu, Yuan Chen, Vanish Talwar, and Dejan Milojevic. Improving Preemptive Scheduling with Application-Transparent Checkpointing in Shared Clusters. In *Proceedings of the 16th International Middleware Conference*, 2015.
- [23] David G. Kendall. Stochastic Processes Occurring in the Theory of Queues and Their Analysis by the Method of the Imbedded Markov Chain. *The Annals of Mathematical Statistics*, 1953.
- [24] Ng Chee-Hock and Soong Boon-Hee. *Queueing Modelling Fundamentals: With Applications in Communication Networks*. John Wiley & Sons, 2008.
- [25] Xinzichen Li. Performance Evaluation of M—M—1 Queueing Models in Cloud Computing Environments. *Highlights in Science, Engineering and Technology CSIC*, 2024.
- [26] Gaurav Jain and Karl Sigman. A Pollaczek-Khintchine Formula for M/G/1 Queues with Disasters. *Journal of Applied Probability*, 1996.
- [27] Ronald W. Wolff. Poisson Arrivals See Time Averages. *Operations Research*, 1982.
- [28] Arcangela Rago, Alessandro Guidotti, Giuseppe Piro, Ernestina Cianca, Alessandro Vanelli-Coralli, Simone Morosi, Giuseppe Virone, Fabrizio Brasca, Martina Troscia, Marina Settembre, Laura Pierucci, Francesco Matera, Mauro De Sanctis, Sara Pizzi, and Luigi Alfredo Grieco. Multi-layer ntn architectures toward 6g: The ita-ntn view. *Computer Networks*, 254, 2024.
- [29] Kai Lai Chung. *Markov Chains With Stationary Transition Probabilities*. Springer-Verlag, 1967.
- [30] Hannah Constantin. Markov Chains and Queueing Theory. In *Proceedings of the 2011 Conference on Queueing Theory and Markov Chains*, 2011.
- [31] Peter W. Glynn. Queueing Theory: Past, Present, and Future. *Queueing Systems*, 2022.
- [32] Mor Harchol-Balter. Open Problems in Queueing Theory Inspired by Data-center Computing. *Queueing Systems*, 2021.
- [33] Michael L. Pinedo. *Planning and Scheduling in Manufacturing and Services*. Springer New York, 2005.

- [34] Essam F. Natsheh. Queue Scheduling Algorithms: A Comparative Analysis. *International Journal of Innovative Computing*, May 2020.
- [35] Swati Lipsa, Ranjan Kumar Dash, Nikola Ivković, and Korhan Cengiz. Task Scheduling in Cloud Computing: A Priority-Based Heuristic Approach. *IEEE Access*, 2023.
- [36] Diletta Olliaro, Marco Ajmone Marsan, Simonetta Balsamo, and Andrea Marin. The Saturated Multiserver Job Queuing Model with Two Classes of Jobs: Exact and Approximate Results. *Performance Evaluation*, 2023.
- [37] Muhammad Waqas, Leonardo Maccari, and Andrea Marin. Finite Capacity Multi-Server Job Systems: A Simulation Study. In *Proceedings of the 38th ECMS International Conference on Modelling and Simulation*, 2024.
- [38] S. Elmougy, S. Sarhan, and M. Joundy. A Novel Hybrid of Shortest Job First and Round Robin with Dynamic Variable Quantum Time Task Scheduling Technique. *Journal of Cloud Computing*, 2017.
- [39] Rasmus V. Rasmussen and Michael A. Trick. Round Robin Scheduling – A Survey. *European Journal of Operational Research*, 2008.
- [40] S. Pemasinghe and S. Rajapaksha. Comparison of CPU Scheduling Algorithms: FCFS, SJF, SRTF, Round Robin, Priority Based, and Multilevel Queuing. In *2022 IEEE 10th Region 10 Humanitarian Technology Conference*, 2022.
- [41] Mehdi Kargahi and Ali Movaghar. A Multiprocessor System with Non-Preemptive Earliest-Deadline-First Scheduling Policy: A Performability Study. *Journal of Industrial and Systems Engineering*, Spring 2007.
- [42] Ehsan Mohammadi and Mehdi Alinaghian. A Preemptive Priority Queue with Preemptions Penalty. *International Journal of Advanced Manufacturing Technology*, 2015.
- [43] Sven Jäger, Guillaume Sagnol, Dominik Schmidt genannt Waldschmidt, et al. Competitive Kill-and-Restart and Preemptive Strategies for Non-Clairvoyant Scheduling. *Mathematical Programming*, 2024.
- [44] U. Kanade. Performance of Work Conserving Schedulers and Scheduling of Some Synchronous Dataflow Graphs. In *Proceedings of the Tenth International Conference on Parallel and Distributed Systems*, 2004.

- [45] Ru Jia, Yun Yang, John Grundy, Jacky Keung, and Li Hao. A Systematic Review of Scheduling Approaches on Multi-Tenancy Cloud Platforms. *Information and Software Technology*, 2021.
- [46] A. Diaz, R. Batista, and O. Castro. Realtss: A Real-Time Scheduling Simulator. In *2007 4th International Conference on Electrical and Electronics Engineering*, 2007.
- [47] Maxime Chéramy, Pierre-Emmanuel Hladik, and Anne-Marie Déplanche. SimSo: A Simulation Tool to Evaluate Real-Time Multiprocessor Scheduling Algorithms. In *5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, 2014.
- [48] R. Valarmathi and T. Sheela. Differed Service Broker Scheduling for Data Centres in Cloud Environment. *Computer Communications*, 2019.
- [49] S. Deepa, K. P. Sridhar, and K. B. Mythili. Scheduling Model for Task Loading in Cloud Data Centres. *Wireless Networks*, 2023.
- [50] M. Safavi and B. Landfeldt. Energy-Efficient Stable and Balanced Task Scheduling in Data Centers. *IEEE Transactions on Sustainable Computing*, 2021.
- [51] Mihaela-Andreea Vasile, Florin Pop, Radu-Ioan Tutueanu, Valentin Cristea, and Joanna Kołodziej. Resource-Aware Hybrid Scheduling Algorithm in Heterogeneous Distributed Computing. *Future Generation Computer Systems*, 2015.
- [52] Doaa M. Abdelkader and Fatma Omara. Dynamic Task Scheduling Algorithm with Load Balancing for Heterogeneous Computing System. *Egyptian Informatics Journal*, 2012.
- [53] S. Gopalt and U. Vajapeyam. Load Balancing in a Heterogeneous Computing Environment. In *Proceedings of the 31st Hawaii International Conference on System Sciences*, 1998.
- [54] Dania A. Shafiq, Nafisa Z. Jhanjhi, and Abdulaziz Abdullah. Load Balancing Techniques in Cloud Computing Environment: A Review. *Journal of King Saud University – Computer and Information Sciences*, 2022.

- [55] Mainak Adhikari, Tarachand Amgoth, and Satish Narayana Srirama. A Survey on Scheduling Strategies for Workflows in Cloud Environment and Emerging Trends. *ACM Computing Surveys*, 2019.
- [56] Essam H. Houssein, Ahmed G. Gad, Yaser M. Wazery, and Ponnuthurai Nagaratnam Suganthan. Task Scheduling in Cloud Computing Based on Meta-heuristics: Review, Taxonomy, Open Challenges, and Future Trends. *Swarm and Evolutionary Computation*, 2021.
- [57] Stewart Robinson. Discrete-Event Simulation: From the Pioneers to the Present, What Next? *Journal of the Operational Research Society*, 2005.
- [58] David Goldsman and Pamela Goldsman. Discrete-Event Simulation. In Margaret Loper, editor, *Modeling and Simulation in the Systems Engineering Life Cycle*. Springer, 2015.
- [59] Peng Zhang. *Advanced Industrial Control Technology*. Elsevier, 2010.
- [60] Jerry Banks, John S. Carson II, Barry L. Nelson, and David M. Nicol. *Discrete-Event System Simulation*. Prentice Hall, 2005.
- [61] Margaret L. Loper, editor. *Modeling and Simulation in the Systems Engineering Life Cycle: Core Concepts and Accompanying Lectures*. Springer, 2015.
- [62] Ahmad Musa and Irfan Awan. Functional and Performance Analysis of Discrete Event Network Simulation Tools. *Simulation Modelling Practice and Theory*, 2022.
- [63] Saba Siraj, Ajay Kumar Gupta, and Rinku Badgujar. Network Simulation Tools Survey. *International Journal of Advanced Research in Computer and Communication Engineering*, 2012.
- [64] T. Arvind. A Comparative Study of Various Network Simulation Tools. *International Journal of Computer Science & Engineering Technology*, 2016.
- [65] Linda Ann Riley. Discrete-Event Simulation Optimization: A Review of Past Approaches and Propositions for Future Direction. In *Proceedings of the 2013 Summer Computer Simulation Conference*, 2013.
- [66] K. Agalinos, S. T. Ponis, E. Aretoulaki, G. Plakas, and O. Efthymiou. Discrete Event Simulation and Digital Twins: Review and Challenges for Logistics. *Procedia Manufacturing*, 2020.

- [67] Rajkumar Buyya, Shashikant Ilager, and Pablo Arroba. Energy-Efficiency and Sustainability in New Generation Cloud Computing: A Vision and Directions for Integrated Management of Data Centre Resources and Workloads. *Software: Practice and Experience*, 2023.
- [68] IMDEA Networks Institute. Datacenter with High Efficiency: Optimizing Organization and Scheduling of Datacenter Resources, 2023.
- [69] Saurabh Sahni, Gaurav Somani, M. Conti, and M. Rajarajan. Security and Privacy in Cloud Computing. In *Proceedings of the 2016 ACM International Symposium on Cloud Computing*, 2016.
- [70] Abhishek Gupta, Paolo Faraboschi, Filippo Gioachin, Laxmikant V. Kale, Richard Kaufmann, Bu-Sung Lee, Verdi March, Dejan Milojicic, and Chun Hui Suen. Evaluating and Improving the Performance and Scheduling of HPC Applications in Cloud. *IEEE Transactions on Cloud Computing*, 2016.
- [71] Harshit Kapoor. Cost and Energy Efficient Job Scheduling Algorithm for Cloud Data Centre. Master’s thesis, The University of Melbourne, 2017.
- [72] Richard Olaniyan and Muthucumar Maheswaran. Research Advances in Cloud Computing. In Sanjay Chaudhary, Gaurav Somani, and Rajkumar Buyya, editors, *Research Advances in Cloud Computing*. Springer Nature Singapore Pte Ltd., 2017.
- [73] Soheil Abbasloo, Yang Xu, and H. Jonathan Chao. To Schedule or Not to Schedule: When No-Scheduling Can Beat the Best-Known Flow Scheduling Algorithm in Datacenter Networks. *Computer Networks*, 2020.
- [74] Afia Bhutto, Aftab Ahmed Chandio, Kirshan Kumar Luhano, and Imtiaz Ali Korejo. Analysis of Energy and Network Cost Effectiveness of Scheduling Strategies in Datacentre. *Cybernetics and Information Technologies*, 2023.
- [75] Hashim Ali, Muhammad Shuaib Qureshi, Muhammad Bilal Qureshi, Ayaz Ali Khan, Muhammad Zakarya, and Muhammad Fayaz. An Energy and Performance Aware Scheduler for Real-Time Tasks in Cloud Datacentres. *IEEE Access*, 2020.
- [76] Banavath Balaji Naik, Dhananjay Singh, and Arun Barun Samaddar. Multi-objective Virtual Machine Selection in Cloud Data Centers Using Optimized Scheduling. *Wireless Personal Communications*, 2021.

- [77] Pamela Isabel Delgado Borda. *Hybrid, Job-Aware, and Preemptive Datacenter Scheduling*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2018.
- [78] Jiamin Li, Hong Xu, Yibo Zhu, Zherui Liu, Chuanxiong Guo, and Cong Wang. Lyra: Elastic Scheduling for Deep Learning Clusters. In *Proceedings of the 18th European Conference on Computer Systems*, 2023.
- [79] Swati Lipsa, Ranjan Kumar Dash, Nikola Ivković, and Korhan Cengiz. Task Scheduling in Cloud Computing: A Priority-Based Heuristic Approach. *IEEE Access*, 2023.
- [80] Xiaojun Bai and Shunfu Jin. Performance Analysis of an Energy-Saving Strategy in Cloud Data Centers Based on a MMAP[K]/M[K]/N1 + N2 Non-Preemptive Priority Queue. *Future Generation Computer Systems*, 2022.
- [81] Qiang Wang, Xinxin Mei, Hai Liu, Yiu-Wing Leung, Zongpeng Li, and Xiaowen Chu. Energy-Aware Non-Preemptive Task Scheduling With Deadline Constraint in DVFS-Enabled Heterogeneous Clusters. *IEEE Transactions on Parallel and Distributed Systems*, 2022.
- [82] Thomas Bartz-Beielstein, Jürgen Branke, Jürgen Mehnen, and Olaf Mersmann. Evolutionary Algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2014.
- [83] Banavath Balaji Naik, Dhananjay Singh, and Arun B. Samaddar. FHCS: Hybridised Optimisation for Virtual Machine Migration and Task Scheduling in Cloud Data Center. *IET Communications*, 2020.
- [84] Elsayed Elsedimy and Fahad Algarni. MOTS-ACO: An Improved Ant Colony Optimiser for Multi-Objective Task Scheduling Optimisation Problem in Cloud Data Centres. *IET Networks*, 2022.
- [85] S. Alangaram and S. P. Balakannan. Optimizing Task Scheduling in Cloud Data Centres with Dynamic Resource Allocation Using Genetic Algorithm. *Journal of Electrical Systems*, 2024.
- [86] C. Peoples, S. McClean G. Parr, P. Morrow, and B. Scotney. Energy Aware Scheduling across Green Cloud Data Centres. *IFIP/IEEE International Symposium on Integrated Network Management*, 2013.

- [87] Fung Po Tso, Simon Jouet, and Dimitrios P. Pezaros. Network and Server Resource Management Strategies for Data Centre Infrastructures: A Survey. *Computer Networks*, 2016.
- [88] Jie Zhao, Maria A. Rodríguez, and Rajkumar Buyya. A Deep Reinforcement Learning Approach to Resource Management in Hybrid Clouds Harnessing Renewable Energy and Task Scheduling. In *Proceedings of the IEEE 14th International Conference on Cloud Computing*, 2021.
- [89] Damián Fernández-Cerero, José A. Troyano, Agnieszka Jakóbik, and Alejandro Fernández-Montes. Machine Learning Regression to Boost Scheduling Performance in Hyper-scale Cloud-Computing Data Centres. *Journal of King Saud University – Computer and Information Sciences*, 2022.
- [90] Ziniu Lin, Chen Li, Lihua Tian, and Bin Zhang. A Scheduling Algorithm Based on Reinforcement Learning for Heterogeneous Environments. *Applied Soft Computing*, 2022.
- [91] Louis G. Birta and Gilbert Arbez. *Modelling and Simulation: Exploring Dynamic System Behaviour*. Springer Cham, 2019.
- [92] Oladosu Oyebisi Oladimeji, Dasola Oyeyiola, Olayanju Oladineji, and Pelumi Oyeyiola. A Comprehensive Survey on Cloud Computing Simulators. *Scientific Journal of Informatics*, 2021.
- [93] I. Bambrik. A Survey on Cloud Computing Simulation and Modeling. *SN Computer Science*, 2020.
- [94] Adityo Anggraito. Multiservers Job Queueing Model for Datacenter Resource Optimization. Master’s thesis, Università Ca’ Foscari Venezia, 2023.
- [95] Ricardo M. Czekster. Tools for Modelling and Simulating the Smart Grid. arXiv preprint, 2020.
- [96] A. Brandwajn and T. Begin. Multi-server Preemptive Priority Queue with General Arrivals and Service Times. *Performance Evaluation*, 2017.
- [97] Tom Van Woensel, Inmaculada Rodríguez, and Nicolaas Vandaele. Buffer and Server Allocation in General Multi-server Queueing Networks. *International Transactions in Operational Research*, 2010.

- [98] Zhixing Lin, Chibiao Liu, and Chia-Hung Wang. Performance Evaluation of Job Queue Management for a Computer Center with Two Heterogeneous Clusters. *Journal of Network Intelligence*, 2021.
- [99] Rakesh Kumar and Sumeet K. Sharma. A Multi-Server Markovian Queueing System with Discouraged Arrivals and Retention of Reneged Customers. *International Journal of Operations Research*, 2012.
- [100] András Varga and Rudolf Hornig. An Overview of the OMNeT++ Simulation Environment. In *1st International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, 2010.
- [101] András Varga. A Practical Introduction to the OMNeT++ Simulation Framework. In Antonio Virdis and Michael Kirsche, editors, *Recent Advances in Network Simulation*. Springer, Cham, 2019.
- [102] Pasquale Imputato and Stefano Avallone. Traffic Differentiation and Multi-queue Networking in ns-3. In *Proceedings of the 2017 Workshop on Ns-3*, 2017.
- [103] M. Çakmak, Z. Albayrak, and C. Torun. Performance Comparison of Queue Management Algorithms in LTE Networks using NS-3 Simulator. *Tehnički vjesnik*, 2021.
- [104] Robert Chang and Vahab Pournaghshband. Simulating Strict Priority Queueing, Weighted Round Robin, and Weighted Fair Queueing with NS-3. *International Journal on Advances in Networks and Services*, 2017.
- [105] A. V. Karthick, E. Ramaraj, and R. Ganapathy Subramanian. An Efficient Multi Queue Job Scheduling for Cloud Computing. In *2014 World Congress on Computing and Communication Technologies*, 2014.
- [106] R. Naim and N. Chaurasia. Analyzing the Concept of Priority Queue for Scheduling in Cloud Scenario. In P.K. Kapur, G. Singh, and S. Panwar, editors, *Advances in Interdisciplinary Research in Engineering and Business Management*. Springer, 2021.
- [107] A. Hussain, M. Aleem, M. A. Iqbal, and M. A. Islam. Investigation of Cloud Scheduling Algorithms for Resource Utilization Using CloudSim. *Computing and Informatics*, 2019.

- [108] Amit Sundas and Surya Narayan Panda. An Introduction of CloudSim Simulation Tool for Modelling and Scheduling. In *2020 International Conference on Emerging Smart Computing and Informatics*, 2020.
- [109] Wanqing You, Kai Qian, and Ying Qian. Hierarchical Queue-Based Task Scheduling. *Journal of Advances in Computer Networks*, June 2014.
- [110] Jan Kopanski. Optimisation of Job Scheduling for Supercomputers with Burst Buffers. arXiv preprint, 2021.
- [111] A. Marin, S. Rossi, and C. Zen. A Matlab Toolkit for the Analysis of Two-Level Processor Sharing Queues. In M. Gribaudo, D.N. Jansen, and A. Remke, editors, *Quantitative Evaluation of Systems*. Springer, Cham, 2020.
- [112] Asad Malik and Samee Khan. Handbook on data centers. In Samee U. Khan and Albert Y. Zomaya, editors, *Handbook on Data Centers*. Springer, New York, NY, 2015.
- [113] Jens Falk et al. NeSTiNg: Simulating IEEE Time-sensitive Networking in OMNeT++. In *Proceedings of the 2019 IEEE International Conference on Network Protocols*, 2019.
- [114] Carl Bachmeier et al. *Network Simulation with OMNeT++*. Springer, 2020.
- [115] Andrzej Chydzinski. On the Stability of Queues with the Dropping Function. *Performance Evaluation*, 2021.
- [116] Andrea Marin, Marco Ajmone Marsan, Michela Meo, and Mauro Sereno. Queuing Models of Links Carrying Streaming and Elastic Services. *Computer Networks*, 2024.
- [117] Ivo Adan and Jacques Resing. *Queueing Systems*. Eindhoven University of Technology, 2015.
- [118] Amir Masoud Rahmani Einollah Jafarnejad Ghomi and Nooruldeen Nasih Qader. Applying Queue Theory for Modeling of Cloud Computing: A Systematic Review. *Concurrency and Computation: Practice and Experience*, 2019.
- [119] Fahd Alhaidari, Atta Rahman, and Rachid Zagrouba. Cloud of Things: Architecture, Applications and Challenges. *Journal of Ambient Intelligence and Humanized Computing*, 2023.

- [120] Ali Belgacem. Dynamic Resource Allocation in Cloud Computing: Analysis and Taxonomies. *Computing*, 2022.
- [121] Li-On Raviv and Amir Leshem. Maximizing Service Reward for Queues with Deadlines. *IEEE/ACM Transactions on Networking*, 2018.
- [122] Qiang Duan. Cloud Service Performance Evaluation: Status, Challenges, and Opportunities – A Survey from the System Modeling Perspective. *Digital Communications and Networks*, 2017.
- [123] Isaac Grosof, Yige Hong, Mor Harchol-Balter, and Alan Scheller-Wolf. The RESET and MARC Techniques, with Application to Multiserver-Job Analysis. *Performance Evaluation*, 2023.
- [124] Diletta Olliaro, Marco Ajmone Marsan, Simonetta Balsamo, and Andrea Marin. The Saturated Multiserver Job Queuing Model with Two Classes of Jobs: Exact and Approximate Results. *Performance Evaluation*, 2023.
- [125] Weina Wang, Qiaomin Xie, and Mor Harchol-Balter. Zero Queueing for Multi-Server Jobs. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2021.
- [126] OMNeT++ Community. Omnet++: Discrete event simulator. <https://omnetpp.org/>, 2024. Accessed: January 12, 2024.
- [127] Ziv Scully, Isaac Grosof, and Mor Harchol-Balter. Optimal Multiserver Scheduling with Unknown Job Sizes in Heavy Traffic. *Performance Evaluation*, 2021.
- [128] Mor Harchol-Balter Isaac Grosof and Alan Scheller-Wolf. Stability for Two-class Multiserver-job Systems. preprint, 2020.
- [129] Anouar El Haji and Sander Onderstal. Trading Places: An Experimental Comparison of Reallocation Mechanisms for Priority Queuing. *Journal of Economics & Management Strategy*, 2019.
- [130] Gary Lee. *Cloud Networking*. Morgan Kaufmann, 2014.
- [131] Roberto Peñaranda, Crispín Gómez, María Engracia Gómez, and Pedro López. XOR-Based HoL-Blocking Reduction Routing Mechanisms for Direct Networks. *Parallel Computing*, 2017.

- [132] Malvinder Bali, Kamali Gupta, Gupta Deepali, Gautam Srivastava, Sapna Juneja, and Ali Nauman. An Effective Technique to Schedule Priority-Aware Tasks to Offload Data on Edge and Cloud Servers. *Measurement: Sensors*, 2023.
- [133] P. G. Harrison and N. M. Patel. *Performance Modelling of Communication Networks and Computer Architectures*. Addison-Wesley, 1992.
- [134] Mert Yildiz and Andrea Baiocchi. Data-Driven Workload Generation Based on Google Data Center Measurements. In *2024 IEEE 25th International Conference on High Performance Switching and Routing*, 2024.
- [135] Arman Shehabi, Sarah Smith, Dale Sartor, Richard Brown, Mats Herrlin, Jonathan Koomey, et al. United States Data Center Energy Usage Report. Technical report, Lawrence Berkeley National Laboratory, 2016.
- [136] Ahmed Fathalla, Kegin Li, and Ahmed Salah. Best-KFF: A Multi-Objective Preemptive Resource Allocation Policy for Cloud Computing Systems. *Cluster Computing*, 2022.
- [137] Amin Isazadeh, Davide Ziviani, and David E. Claridge. Global Trends, Performance Metrics, and Energy Reduction Measures in Datacom Facilities. *Renewable and Sustainable Energy Reviews*, 2023.
- [138] Andrei V. Sleptchenko, Arie van Harten, and Matthieu C. van der Heijden. Analyzing Multi-Class, Multi-Server Queueing Systems with Preemptive Priorities. Technical report, Technische Universiteit Eindhoven, BETA, 2002.
- [139] Andrei Sleptchenko, Aart van Harten, and Matthieu van der Heijden. An Exact Solution for the State Probabilities of the Multi-Class, Multi-Server Queue with Preemptive Priorities. *Queueing Systems*, 2005.
- [140] Mulya Agung, Yuta Watanabe, Henning Weber, Ryusuke Egawa, and Hiroyuki Takizawa. Preemptive Parallel Job Scheduling for Heterogeneous Systems Supporting Urgent Computing. *IEEE Access*, 2021.
- [141] Seokjun Lee, Chesoong Kim, Sergey Dudin, and Olga Dudina. Multi-Server Queueing Model with Many Types of Priority Customers. *Industrial Engineering & Management Systems*, 2023.

- [142] Ebrahim Jafarpour Ghomi, Amir Masoud Rahmani, and Nematollah Navab Qader. Applying Queue Theory for Modeling of Cloud Computing: A Systematic Review. *Concurrency and Computation: Practice and Experience*, 2019.
- [143] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale Cluster Management at Google with Borg. In *Proceedings of the 10th European Conference on Computer Systems*, 2015.
- [144] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E. Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: The Next Generation. In *Proceedings of the 15th European Conference on Computer Systems*, 2020.