Finite Capacity Multi-Server Job Systems: A Simulation Study

Muhammad Waqas[1], Leonardo Maccari[1], and Andrea Marin[1]

[1]Università Ca' Foscari Venezia, Venice, Italy

ABSTRACT

Cloud computing has revolutionized how computational resources are accessed and utilized. However, the dynamic nature of the cloud computing environment, which is characterized by a variety of resource types and capabilities presents challenges for managing the workload and ensuring the quality of service. The selection and implementation of queueing policies can have a major impact on the efficiency of the cloud environment, and thus on the quality of service 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, our 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 proposed 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 First-Come-First-Serve and Priority queueing. The result revealed that assigning priority to jobs with longer service times and larger resource demands has a positive impact on small jobs as well.

KEYWORDS

Simulation, Modeling, Finite Capacity Queues, Dropping.

INTRODUCTION

Cloud computing is a way to access a variety of computing resources, like networks, servers, storage, and applications, over the internet. These resources can be quickly set up and taken down with little effort. Cloud computing can make the basic IT resources into a resource pool that can be freely scheduled, to realize the on-demand allocation, and to provide customer services Buyya et al. (2011)Gong et al. (2010)Dillon et al. (2010). In recent years, cloud computing has become a key part of how we use technology, offering a flexible and scalable way to access computing power Einollah Jafarnejad Ghomi and Qader (2019). Nevertheless, the diverse nature of the cloud computing environment, with its wide range of resource types and capabilities, presents challenges for managing workloads and ensuring quality of service. At present, the cloud computing field is mainly facing the following problems: (a) Resource service scheduling, (b) Task segmentation, (c) Network transmission, and (d) Real-world task application. Besides 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, see, e.g., Alhaidari et al. (2023) Belgacem (2022) and Raviv and Leshem (2018) for some recent works on the field. Improving the reliability of cloud systems can be achieved by carefully analyzing and modeling key elements such as task scheduling, service time, the waiting period for these services, and the likelihood of hardware and software failures Einollah Jafarnejad Ghomi and Qader (2019).

Understanding the performance metrics of different queueing policies in a scalable cloud computing environment is essential for both cloud service providers and consumers Duan (2017). In this work, we have considered a simple abstraction of a cloud infrastructure based on the Multi-Server Job Queueing Model (MJQM). With respect to other research streams in the field, we have employed the Finite Capacity Queue (FCQ) rather than an infinite one as in Wang et al. (2021), Grosof et al. (2023), Olliaro et al. (2023). Queueing models have been extensively applied in resource-sharing systems like production, communication, and computer systems.

In a nutshell, the MJQM consists of a set of resources and various classes of jobs. The class determines the stochastic duration of the job's service time and the deterministic amount of resources required to be served. Jobs that cannot find enough resources to be served must wait for them in the queue.

Several policies have been defined to address the problem of how to allocate the resources to the jobs in the queue that vary according to the information available to the scheduler. In this work, we assume that the scheduler knows only the amount of resources required by each job (and not, e.g., the service time).

The contributions of the proposed work are summarized as follows:

1. **Open Source Simulation Model**: We designed a simulator based on OMNeT++[1] and made the code open source and available on GitHub (see Section Availability).

[1]https://omnetpp.org

Communications of the ECMS, Volume 38, Issue 1,
Proceedings, ©ECMS Daniel Grzonka, Natalia Rylko,
Grazyna Suchacka, Vladimir Mityushev (Editors) 2024
ISBN: 978-3-937436-84-5/978-3-937436-83-8(CD) ISSN 2522-2414

2. **Investigation of the dropping probability in a simple scenario**: We consider two widely applied scheduling policies without preemption: First-In-First-Out (FIFO) and a discipline giving hard priority to larger classes of jobs. We investigate the dropping probability due to the finite capacity buffer in a scenario consisting only of two classes. Despite the simplicity of the configuration, we unveil some unexpected (and, in our view, counter-intuitive) behaviors of the performance indices and provide an explanation.

The rest of the paper is organized as follows: Section provides the necessary background for employing various queueing disciplines in the FCQ to evaluate the relationship between performance and the underlying scheduling policy. Section discusses general problem settings, including types of jobs, their arrival probabilities, job service times, and other model characteristics like resource and buffer capacities. Section 4 presents the insights derived from the study, comparing performance metrics under different scenarios. Finally, Section 5 concludes the paper, summarizing key findings and discussing potential avenues for future research.

LITERATURE REVIEW

Nowadays, cloud computing is widely used as an effective and efficient way to integrate resources and services, transforming hardware and software resources into common goods and utilities as needed by end users. This immense growth has not only increased the number and diversity of applications but also intensified the frequency of communications. Consequently, this rapid increase has made the task of job scheduling and resource allocation even more crucial, as they now play a pivotal role in meeting the service level agreements. So to improve the performance of the server, minimizing the completion time and cost it is very necessary to schedule the tasks in the cloud optimally. In Jena (2017), the authors introduced a scheduling algorithm optimized for both energy efficiency and processing time, validating their simulation model through comparisons with existing algorithms to demonstrate superior effectiveness. The study done by Ding et al. (2020) proposed a learning-based approach to task scheduling, tackling the dual challenges of energy consumption and the improvement of cloud services' efficiency and responsiveness. The study in Wu et al. (2018) developed a method for virtual machine allocation that ensures the reliability and timely completion of workflow applications in cloud services.

The studies in Terekhov et al. (2014) and Kebarighotbi and Cassandras (2011) have utilized the FIFO discipline to integrate queueing theory with scheduling for dynamic system efficiency and to optimize profit in cloud services while adhering to service level agreements. Several latest studies included Terekhov et al. (2014),Marin and Rossi (2020), and Lee et al. (2022) collectively underscore the versatility of queue prioritization in enhancing system efficiency and fairness across different queueing environments, from optimizing resource allocation in stochastic models to ensuring unbiased server distribution and prioritizing tasks based on size. However, to the best of our knowledge, there have been no studies reporting the performance evaluation of FCQ using different queueing policies.

With respect to these works, our main focus is on the analysis of the dropping probabilities of finite capacity systems. The importance of considering finite capacity systems is due to the fact that, besides in some simplified scenarios, the stability condition of MJQM is not known. Clearly, a finite capacity system is unconditionally stable but the price to pay is the dropping of some jobs arriving when the buffer is full.

THE QUEUEING MODEL

One of the prominent characteristics of scheduling in data centers is the the relatively low utilization (see, e.g., Scully et al. (2021)) that they can achieve under FIFO policy. In order to investigate the impact of this phenomenon on the dropping probabilities in finite capacity systems, we resort the an MJQM with the following characteristics: (i) we have N identical servers; (ii) jobs arrive at the system belonging to one of two classes, 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 scheduler decides which job to fetch from the buffer whenever there are free computational resources. The queueing system has a finite capacity buffer shared by two classes of jobs. In the following subsection, we describe the FIFO and the Priority scheduling that we compare using our simulator.

FIFO Scheduler

This policy operates on a First-Come, First-Served basis, where the server allocates available resources to jobs in the order of their arrival Isaac Grosof and Scheller-Wolf (2020). Consequently, when the job queue buffer reaches its maximum capacity, any new jobs arriving are dropped until space becomes available. However, the FIFO scheduling policy may encounter significant performance issues in scenarios where available resources are insufficient for the next job in the queue Haji and Onderstal (2019). In such cases, the affected job, along with any subsequent ones 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 Head-of-Line (HOL) blocking, occurs when the first job in the queue,

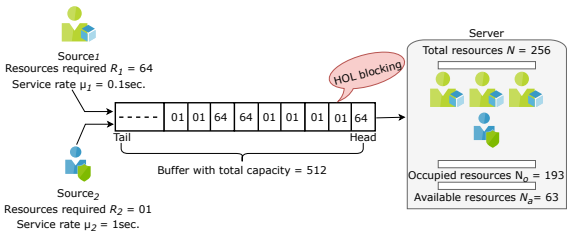


Figure 1: FIFO Queueing with HOL blocking

cannot be served due to insufficient resources. As a result, all subsequent jobs, even those requiring minimal resources, are blocked (Lee, 2014, Ch. 3) Olliaro et al. (2023).

The implications of HOL blocking in such a system are multifaceted: It reduces the maximum throughput by interrupting job processing, increases latency as jobs are uniformly delayed regardless of their size, leads to resource under-utilization due to idling while waiting for the HOL job to be served, and increases the probability of job drops when the buffer reaches its maximum capacity in systems with finite buffer capacities Peñaranda et al. (2017). The HOL blocking in FIFO is influenced by specific factors such as the service time required to finish a job and the resource requirement of big jobs at the head of the queue. As shown in Figure 1, the FIFO queueing model may experience the HOL blocking where jobs from Source$_1$ (requiring 64 cores) and Source$_2$ (requiring 1 core) are processed in the order they arrive. HOL blocking is indicated, showing the delay and under-utilization caused when large jobs from Source$_1$ at the HOL require more resources than are currently available, resulting in subsequent small jobs having to wait their turn.

Hard Priority Scheduler

This policy diverges from FIFO by focusing on high-priority jobs, which, in our case, are the 'Big Jobs'. When resources are available, the server prioritizes these high-priority jobs, reducing their waiting time Bali et al. (2023). This means that if even one 'Big Job' is in the queue awaiting resources, 'Small Jobs' will not receive the service. As for 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 both the arrival frequency and the service time of big jobs. It can lead to prolonged delays for small jobs if big jobs are frequent and consume most of the resources. Figure 2, depicts the Priority queueing model, where Source$_1$'s big jobs are given precedence over Source$_2$'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.

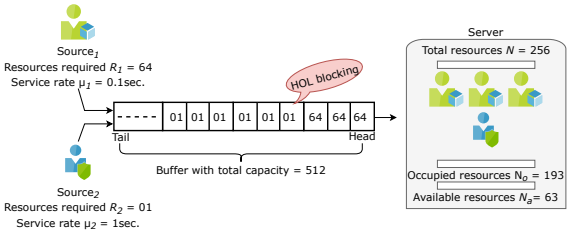


Figure 2: Priority Queueing with HOL blocking

THE SIMULATION MODEL

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

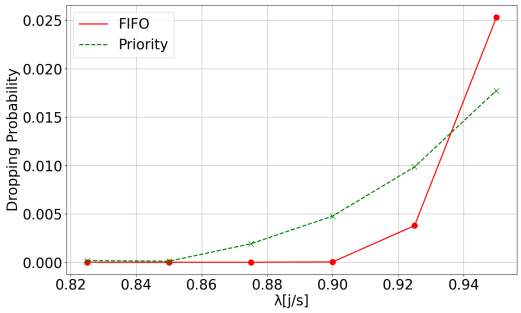
- λ: jobs arrival rate measured in jobs per second;
- μ_s: service rate for small jobs, measured in jobs per second;
- μ_b: service rate for big jobs, measured in jobs per second;
- p: probability that a job belongs to the class of small jobs;
- T: number of servers required by the class of big jobs;
- N: total number of servers available;
- B: available positions in the buffer excluding the jobs in service. Notice that, in the buffer, small and big jobs occupy only one position.
- Scheduling policy: the policy used by the system to decide who is the next job to be served.

Once the 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 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 with an M1 chip.

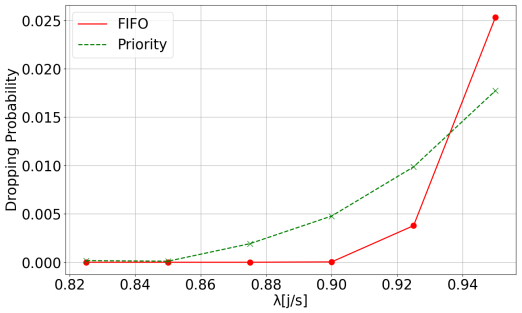
RESULTS

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

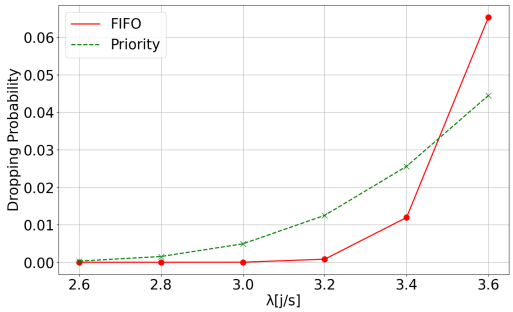
Notice that the following proposition holds:



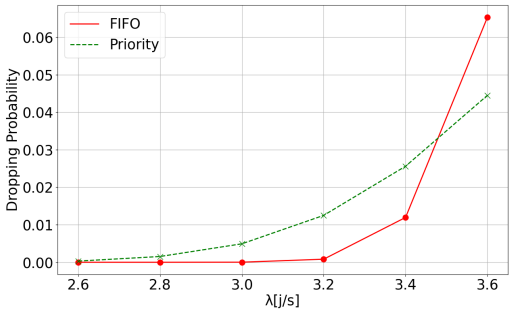
(a) Big Job dropping probability for $p = 0.6$



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

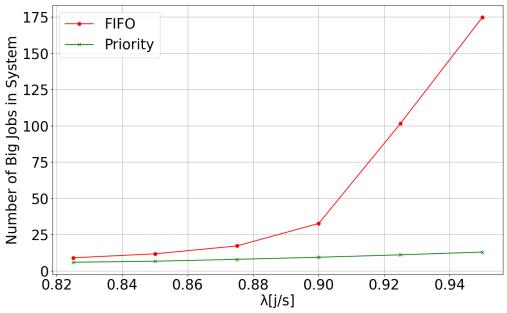


(c) Big Job dropping probability for $p = 0.9$

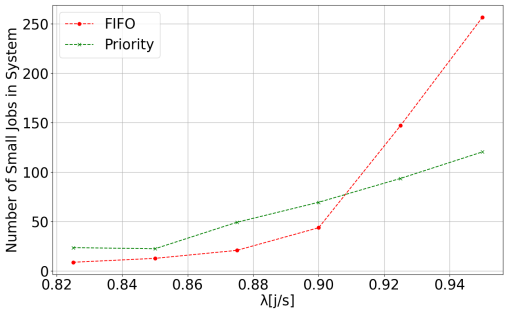


(d) Small Job dropping probability for $p = 0.9$

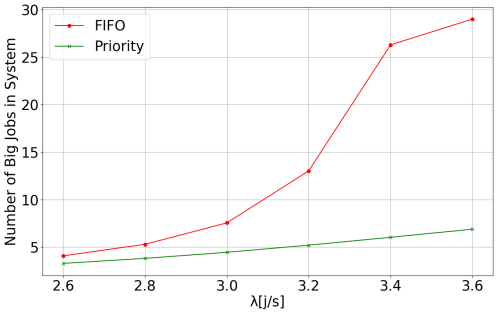
Figure 3: Dropping probability of Big and Small jobs as a function of λ



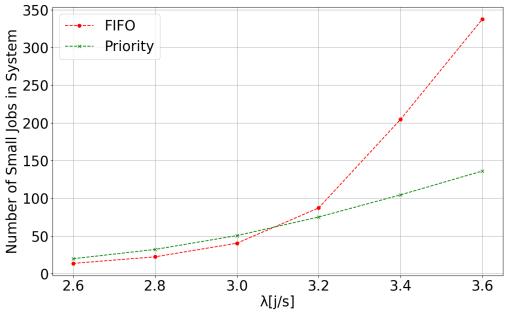
(a) Expected n. of Big Jobs in the system for $p = 0.6$



(b) Expected n. of Small Jobs in the system for $p = 0.6$

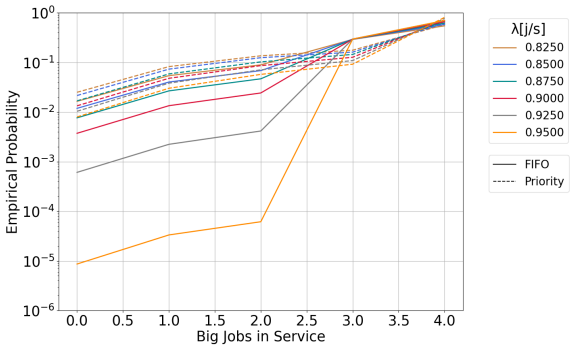


(c) Expected n. of Big Jobs in the system for $p = 0.9$

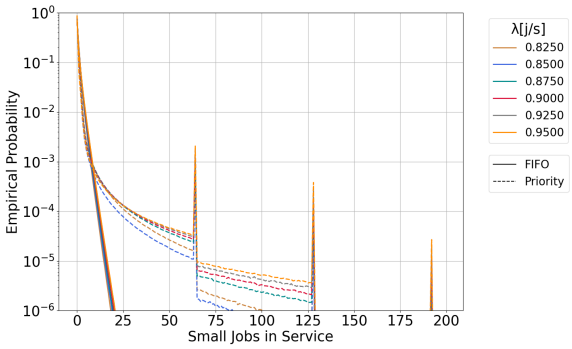


(d) Expected n. of Small Jobs in the system for $p = 0.91$

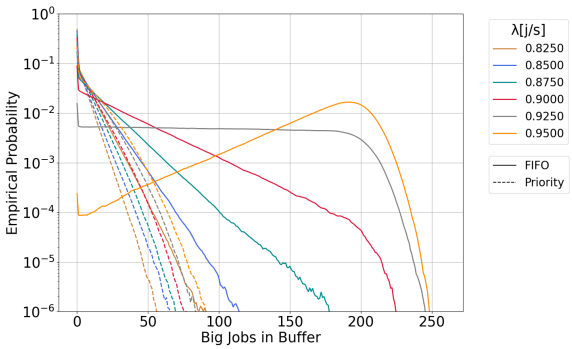
Figure 4: Expected occupancy in the system as function of λ



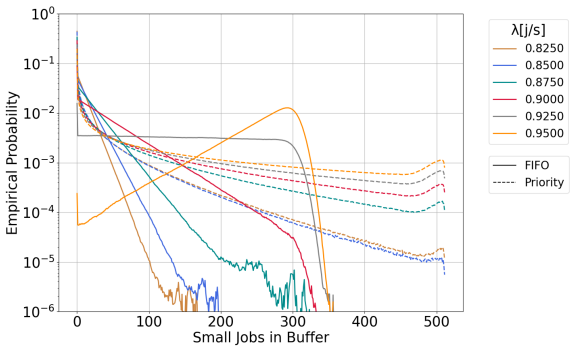
(a) Distribution of the Big Jobs in service $p = 0.6$



(b) Distribution of the Small Jobs in service with $p = 0.6$

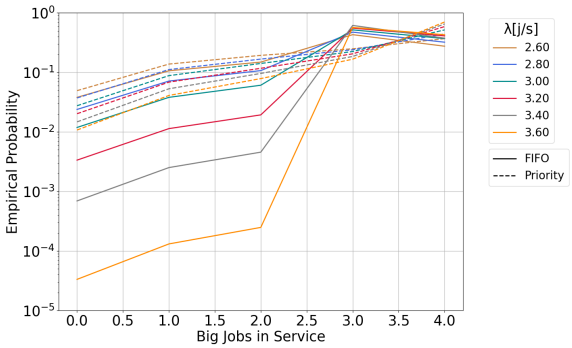


(c) Distribution of the Big Jobs in the buffer with $p = 0.6$

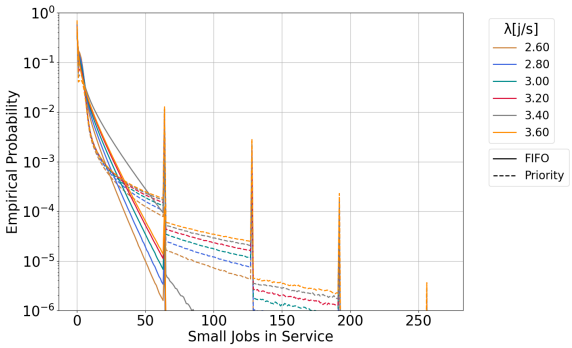


(d) Distribution of the Small Jobs in the buffer with $p = 0.6$

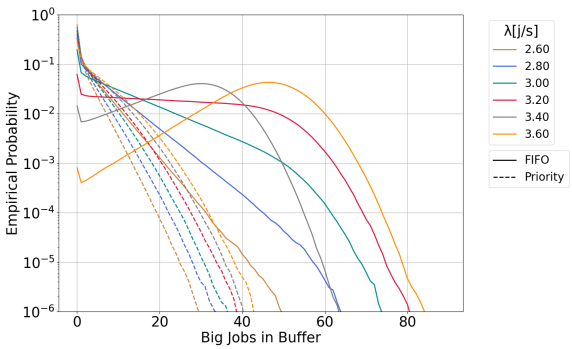
Figure 5: Jobs in service and in the buffer with $p = 0.6$.



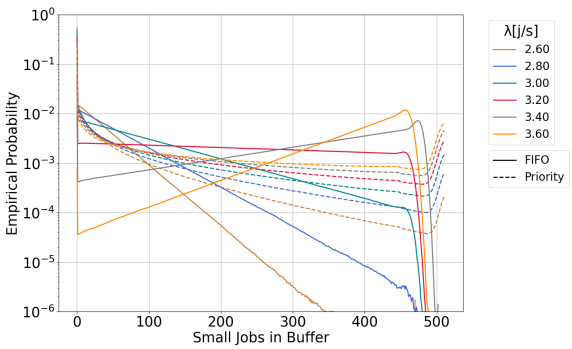
(a) Big Jobs in service for $p = 0.9$



(b) Small Jobs in service for $p = 0.9$



(c) Big Jobs in the buffer for $p = 0.9$



(d) Small Jobs in the buffer for $p = 0.9$

Figure 6: Jobs in service and in the buffer with $p = 0.9$.

Table 1: System Parameters

Buffer Capacity B	512
Service Rate μ_b, μ_s [j/s]	0.1, 1.0
Available Servers N	256
Size of big jobs T	64
Repetitions	30
Sim-Time Limit	5000000s

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/small jobs are independent Poisson processes. Now, a dropping occurs at the arrival of a job in the saturated system. Recall that by the *Poisson Arrivals See Time Averages* (PASTA) property, the customer arriving according to a Poisson process sees the state of the system seen by a random observer Harrison and Patel (1992). Therefore, both big and small jobs see, at their arrival epoch, the same statistics on the system state and share the same dropping probability.

Hereafter, we use the parameterization of the model shown in Table 1.

Figure 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.

First of all, notice 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 resources when the 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 a 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 load, 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).*

The average occupancy at the system for Big and Small jobs is shown in Figures 4a, 4c and 4b, 4d, respectively. Recall that the average occupancy is strictly connected to the expected response time.

Insight 3 *Although 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 w.r.t. to FIFO. This is well supported by the intuition. The situation is more intriguing for what concerns the small jobs. In fact, we notice that under moderate load ($\lambda = 3.6$ j/s) the average occupancy of small jobs is lower under Priority than 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.*

Figures 5a and 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 the small jobs in service in figure 5b depict the difference between the resources required by big jobs and small jobs which is 64 vs. 1 in our case.

Figures 5c and 5d show the buffer state as a function of arrival rate. 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.

Furthermore, figures 6a and 6b show the probability of the average number of jobs in service when the arrival probability of big and small jobs are in the ratio of 10%-90%. 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.

Similar observations can be drawn from Figures 6c and 6d considering the scenario with $p = 0.1$.

CONCLUSION

In this paper, we propose a simulation study of the Multiserver-Job Queueing Model with finite capacity and employing two different scheduling disciplines. 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 requests 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. Additionally, our study was constrained by the consideration of only a single type of resource, with every incoming job requiring the same resource type.

Our simulations show some counter-intuitive effects of the 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. Future works include trying our simulator on real-world traces with numerous sources and resources types and implementing other service policies.

SIMULATOR AVAILABILITY

The complete source code and evaluation results for our models, including FIFO and Priority queueing policies, are publicly accessible in 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: FCQ Repository on GitHub.

REFERENCES

Fahd Alhaidari, Atta Rahman, and Rachid Zagrouba. Cloud of things: architecture, applications and challenges. *J. Ambient Intell. Human Comput*, 14:5957–5975, 2023.

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*, 26:100670, 2023.

Ali Belgacem. Dynamic resource allocation in cloud computing: analysis and taxonomies. *Computing*, 104:681–710, 2022.

Rajkumar Buyya, James Broberg, and Andrzej Goscinski, editors. *Cloud Computing: Principles and Paradigms*. John Wiley and Sons, Inc, Mar 2011.

Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: Issues and challenges. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 27–33, 2010.

Ding Ding, Xiaocong Fan, Yihuan Zhao, Kaixuan Kang, Qian Yin, and Jing Zeng. Q-learning based dynamic task scheduling for energy-efficient cloud computing. *Future Generation Computer Systems*, 108:361–371, 2020.

Qiang Duan. Cloud service performance evaluation: status, challenges, and opportunities – a survey from the system modeling perspective. *Digital Communications and Networks*, 3(2):101–111, 2017.

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*, 31(17):e5186, 2019.

Chunye Gong, Jie Liu, Qiang Zhang, Haitao Chen, and Zhenghu Gong. The characteristics of cloud computing. In *2010 39th International Conference on Parallel Processing Workshops*, pages 275–279, 2010.

Isaac Grosof, Yige Hong, Mor Harchol-Balter, and Alan Scheller-Wolf. The RESET and MARC techniques, with application to multiserver-job analysis. *Perform. Evaluation*, 162:102378, 2023.

Anouar El Haji and Sander Onderstal. Trading places: An experimental comparison of reallocation mechanisms for priority queuing. *J. Econ. Manage. Strat.*, 28:670–686, 2019.

P. G. Harrison and N. M. Patel. *Performance Modelling of Communication Networks and Computer Architectures*. Addison-Wesley, 1992.

Mor Harchol-Balter Isaac Grosof and Alan Scheller-Wolf. Stability for two-class multiserver-job systems. https://arxiv.org/abs/2010.00631, 2020.

R. K. Jena. Energy efficient task scheduling in cloud environment. *Energy Procedia*, 141:222–227, 2017.

Ali Kebarighotbi and Christos G. Cassandras. Optimal scheduling of parallel queues using stochastic flow models. *Discrete Event Dyn. Syst.*, 21: 547–576, 2011.

Gary Lee. *Switch Fabric Technology*, pages 37–64. Morgan Kaufmann, 2014.

Seokjun Lee, Alexander Dudin, Olga Dudina, and Chesoong Kim. Analysis of a priority queueing system with the enhanced fairness of servers scheduling. *Journal of Ambient Intelligence and Humanized Computing*, 2022.

Andrea Marin and Sabina Rossi. A queueing model that works only on the biggest jobs. In *Proc. of Computer Performance Engineering (EPEW)*, volume 12039 of *Lecture Notes in Computer Science*, 2020.

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*, 162:102370, 2023.

Roberto Peñaranda, Crispín Gómez, María Engracia Gómez, and Pedro López. XOR-based HoL-blocking reduction routing mechanisms for direct networks. *Parallel Computing*, 67:57–74, 2017.

Li-On Raviv and Amir Leshem. Maximizing service reward for queues with deadlines. *IEEE/ACM Transactions on Networking*, 26(5):2296–2308, 2018.

Ziv Scully, Isaac Grosof, and Mor Harchol-Balter. Optimal multiserver scheduling with unknown job sizes in heavy traffic. *Perform. Evaluation*, 145:102150, 2021.

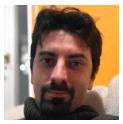
Daria Terekhov, Tony T. Tran, Douglas G. Down, , and J. Christopher Beck. Integrating queueing theory and scheduling for dynamic scheduling problems. *Journal of Artificial Intelligence Research*, 50:535–572, 07 2014.

Weina Wang, Qiaomin Xie, and Mor Harchol-Balter. Zero queueing for multi-server jobs. *Proc. ACM Meas. Anal. Comput. Syst.*, 5(1):07:1–07:25, 2021.

Tingming Wu, Haifeng Gu, Junlong Zhou, Tongquan Wei, Xiao Liu, and Mingsong Chen. Soft error-aware energy-efficient task scheduling for workflow applications in dvfs-enabled cloud. *Journal of Systems Architecture*, 84, 2018.

AUTHOR BIOGRAPHIES

MUHAMMAD WAQAS completed his MS(SE) from COMSATS University, Islamabad, Pakistan. Currently, he is a Ph.D. Scholar at the Department of Environmental Sciences, Informatics and Statistics, Università Ca' Foscari Venezia, Venice, Italy. His research interests include routing algorithms in distributed systems, social networks, and high-performance computing systems. He can be reached using `muhammad.waqas@unive.it`



LEONARDO MACCARI, is associate professor at Università Ca' Foscari Venezia, Italy (`www.dais.unive.it/ maccari`). His work focuses on enhancing distributed wireless networks and network security. He has been working on wireless mesh networks, mobile networks both in the academia and in industrial projects. He is with the Department of Environmental Sciences, Informatics and Statistics (DAIS) at Ca' Foscari since 2019.

ANDREA MARIN, is professor of Computer Science at Università Ca' Foscari Venezia. His focus is on system performance and reliability. He has worked on queueing models, stochastic process algebras, and stochastic Petri nets for the performance evaluation of computer and telecommunication systems His email address is `marin@unive.it` and his web page can be found at `www.dsi.unive.it/~marin/`.