The Impact of Service Demand Variability on Data Center Performance

Diletta Olliaro, Adityo Anggraito, Marco Ajmone Marsan, *Life Fellow, IEEE*, Simonetta Balsamo, and Andrea Marin, *Senior Member, IEEE*

***Abstract*—Modern data centers feature an extensive array of cores that handle quite a diverse range of jobs. Recent traces, shared by leading cloud data center enterprises like Google and Alibaba, reveal that the constant increase in data center services and computational power is accompanied by a growing variability in service demand requirements. The number of cores needed for a job can vary widely, ranging from one to several thousands, and the number of seconds a core is held by a job can span more than five orders of magnitude. In this context of extreme variability, the policies governing the allocation of cores to jobs play a crucial role in the performance of data centers. It is widely acknowledged that the First-In First-Out (FIFO) policy tends to underutilize available computing capacity due to the varying magnitudes of core requests. However, the impact of the extreme variability in service demands on job waiting and response times, that has been deeply investigated in traditional queuing models, is not as well understood in the case of data centers, as we will show. To address this issue, we investigate the dynamics of a data center cluster through analytical models in simple cases, and discrete event simulations based on real data. Our findings emphasize the significant impact of service demand variability, both in terms of requested cores and service times, and allow us to provide insight for enhancing data center performance. In particular, we show how data center performance can be improved thanks to the control of the interplay between service and waiting times through the assignment of cores to jobs.**

***Index Terms*—Data center, performance, analysis, simulation, multiserver job queuing model.**

I. Introduction

THE commercial success of cloud computing services, a market of over half trillion dollars in 2023, has produced skyrocketing numbers of service requests, resulting in soaring system workloads that have been matched by a tremendous surge in computational power. The study of cloud data center workloads reveals that service requests increasingly exhibit extreme variability in terms of the required number of cores and job execution times. For example, in [1], the authors analyze recent Google Borg scheduler traces and show that job demands, in terms of both the number of cores and execution times, vary over several orders of magnitude. Furthermore, execution times often positively correlate with core demands. In such a context of extreme variability, the policies to schedule jobs over the available cores are extremely important for the efficient use of the data center resources, and models (both analytical and simulation) are needed for the prediction of the performance of the possible scheduling policies under the expected job workload. Unfortunately, simulation is costly due to the very long simulation runs that are necessary to adequately capture the (possibly significant) influence of events happening with very low probability, and analysis is difficult. This is because the complexities of the data center operations render inadequate the traditional queuing models often used for the performance evaluation of computing systems. Those models are such that service is provided by a service station comprising one or more servers, but each service only requires one server to work for a variable amount of time. This means that service is one-dimensional, and the only variability in service concerns the time dimension. To adequately model the data centers of today, we instead need at least a *two-dimensional* description of service, thus capturing the fact that the execution of a task requires a variable number of cores for a variable amount of time.

This leads to a rather new construct in queuing theory, that is known under the name *multiserver job queuing model* (MJQM). The MJQM, [2], [3], [4], was introduced to study the behavior of jobs to be executed in a computing facility comprising a large number of machines, as a function of their service requests in terms of the number of requested cores and service duration. The most evident characteristic of the MJQM, which makes it quite different from traditional queuing models, is the fact that it is not work-conserving[1] under the traditional First-In First-Out (FIFO) queuing policy. This is because a service request can be accepted into service only if a sufficient number of service units is free. Otherwise, the service request waits at the head of the waiting line, until the required number of service units becomes free. This generates *head of the line* (HOL) blocking and forces some servers to remain idle while the waiting line comprises customers that could be put in service (see the example in Fig. 1).

Not much is known about the MJQM performance under the FIFO policy (see, e.g., [4], [6], [7]), because the addition of a second dimension to service, and the consequent loss of work conservation makes the model much more difficult to study with analytical approaches with respect to traditional queues. Investigating this specific kind of systems under a FIFO scheduling

Received 8 March 2024; revised 4 September 2024; accepted 1 November 2024. Date of publication 14 November 2024; date of current version 12 December 2024. This work was supported by INdAM-GNCS. Recommended for acceptance by E. Smirni. *(Corresponding author: Diletta Olliaro.)*

Diletta Olliaro, Adityo Anggraito, Simonetta Balsamo, and Andrea Marin are with the Universitá Ca' Foscari Venezia, 30123 Venezia, Italy (e-mail: diletta.olliaro@unive.it).

Marco Ajmone Marsan is with the IMDEA Networks Institute, 28918 Leganés, Spain, and also with Politecnico di Torino, 10129 Torino, Italy.

Digital Object Identifier 10.1109/TPDS.2024.3497792

[1]A scheduling discipline is said to be *work-conserving* if no work is created or wasted within the service room, i.e.: (i) all servers are busy serving jobs whenever there are jobs waiting, and (ii) the work done serving a job is never invalidated [5].

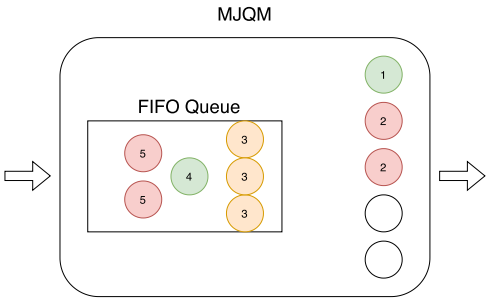


Fig. 1. Example of HOL blocking due to FIFO in the MJQM. In this sketch, the data center has 5 cores, 3 occupied by jobs 1 and 2. Job 3 is the next to be served, but cannot find enough free cores. The FIFO policy imposes jobs 4 and 5 to wait in the queue even if one of them could be served.

policy assumption may seem simplistic in comparison to the complex scheduling algorithms in use in today's data centers. However, FIFO remains prevalent in the management of batch jobs and is integral to specific frameworks such as [8]. Additionally, many real-world schedulers operate in a near-FIFO manner (e.g., [9], [10], [11]), especially when ensuring fairness among different classes of jobs is a critical requirement. Therefore, to effectively study more complex systems, it is essential to first establish a thorough understanding of the FIFO scenario.

In the literature, there is widespread consensus on the fact that the variability in resource demands of jobs can potentially lead to significant underutilization of the data center's resources (see, e.g., [1], [4], [6], [12], [13], [14]) when the FIFO scheduling policy is used, mainly because of HOL blocking. Conversely, policies allowing smaller jobs to overtake the HOL job can result in starvation or significant unfairness toward larger jobs. This effect is undesirable, despite the potential for maximizing utilization, as larger jobs typically contribute more significantly to the data center operator's profitability.

A. Contributions

In this paper, we investigate the role of variability in job service times combined with variability in resource demands. Note that we use the term variability, intentionally avoiding terms like variance or standard deviation, because, as we will see, the impact of variability concerns not only second order moments. Our findings are very surprising and bear significant practical relevance. Indeed, traditional queuing theory models show that the service time variance has an impact on the average system response time, not on the system stability region. In other words, traditional models imply that a server's maximum work capacity depends solely on its speed and the average service demand of jobs. Through a simple analytical counter-example and a simulation study of Google data center traces, we demonstrate that this property does not hold for the MJQM. Our key result is that *in multiserver job systems, the variability in service demands influences both system response time and stability region*. This novelty is explained through simple examples and stated in Remark 1 later in this paper.

The second contribution of this work involves proposing a method to alleviate the resource waste stemming from the variability in the job workload. At first glance, this appears challenging since service times are generally not known in advance, and controlling their variability is often impractical. However, we know in advance the service requests in terms of number of cores. The method we propose entails defining a limited number of core assignment templates and categorizing jobs into these templates. Some jobs will receive more cores than requested, while others will receive fewer. In the former case, there is a potential waste of resources since a job may not be designed to exploit a certain level of parallelism. In the latter case, multiple tasks must share the same core, leading to longer service times. Despite these seemingly unfavorable aspects, optimizing core allocation reduces fragmentation in demands and, consequently, the likelihood of HOL blocking. As a result, *by enhancing data center utilization, we demonstrate that, under moderate and heavy load conditions, the expected response time and the expected response time conditional on the aggregated resource request with our proposed approach are significantly lower than those measured without core assignment templates.*

B. Methodology

We study the impact on performance of the variability in jobs' resource demands, in terms of both the number of cores and core holding times. Our study is conducted with analytical models in simple settings (the only ones for which analytical results are available) and with simulation models using data derived from measurements of real cloud data centers.

To begin with, it is important to clarify that the number of requested cores leads to the job clustering into classes. Thus, a job belongs to a class that reflects its requested number of cores. Each job also has its own service time, and the collection of service times of jobs in a class reflects the probability density function (pdf) of service time for the job class. The class service time pdf can be, for example, exponential (but we will explore other cases as well). The mixing of jobs of different classes into service leads to an overall service time pdf, that, for example, in the case of 3 classes, each with its exponential service time distribution, leads to a Hyperexponential-3 pdf for the overall service time (since each job service with given probabilities belongs to one of 3 exponential pdfs). We will explore the impact on performance of the variability in the number of cores requested by job classes, of the class service time pdf, and of the overall service time pdf.

In Section II, we use and extend our analytical models in [4], to show that the stability region of even simple MJQM settings depends on the distribution of job service times.

In the rest of the paper, we use a measurement-based simulation model to first illustrate the impact on performance of job service times and core allocations and then to propose approaches to improve performance through counterintuitive approaches for core allocation.

Our simulations are based on the measurements reported in [1]. Starting from [1], the authors of [15] extracted the number of used cores and average core holding times for two clusters of machines in a Google data center. It is worth noting that, according to the terminology used in the job management software in Google data centers (namely, Borg), these clusters are referred to as *cells*. The number of used cores defines the job class, and the existing number of classes is extremely high. We

TABLE I
ORIGINAL GOOGLE DATA FOR CELLS A AND B

i	$T_i^{(A)}$	$p_i^{(A)}$	$\tau_i^{(A)}[s]$	$T_i^{(B)}$	$p_i^{(B)}$	$\tau_i^{(B)}[s]$
1	1	9.4e-1	2.1e+0	1	8.4e-1	6.7e+0
2	2	2.1e-2	3.1e+0	2	5.0e-2	2.2e-1
3	3	3.7e-3	1.2e+1	3	6.5e-3	5.2e+0
4	4	2.0e-3	6.4e+1	4	2.4e-2	3.2e+0
5	5	2.9e-3	1.2e+1	5	8.8e-4	1.0e+2
6	6	6.0e-4	1.7e+1	6	1.5e-3	1.2e+2
7	7	2.5e-4	8.0e+1	9	8.6e-5	5.9e+0
8	8	9.3e-5	3.9e+2	10	5.7e-3	1.4e+1
9	9	4.8e-3	5.5e-1	11	3.4e-5	3.9e+3
10	10	6.8e-4	8.6e+1	14	4.0e-5	2.3e+2
11	11	9.6e-5	2.6e+0	15	3.9e-4	1.3e+1
12	14	9.8e-4	1.4e-1	16	7.8e-5	1.4e+4
13	18	9.2e-3	4.5e-1	20	6.9e-4	1.0e+2
14	20	2.9e-4	9.9e+1	30	2.4e-4	7.4e-1
15	24	6.0e-5	2.1e+1	35	2.2e-4	2.1e-1
16	27	5.9e-3	3.1e-1	38	5.8e-5	7.3e+1
17	28	1.9e-3	9.7e-2	50	7.2e-4	5.3e+1
18	30	5.3e-5	2.0e+1	98	8.8e-5	8.1e-1
19	38	1.1e-3	2.2e-1	99	7.6e-4	5.2e+0
20	40	2.6e-3	1.4e+0	100	6.9e-2	3.1e+0
21	42	9.4e-4	1.6e-1	120	3.6e-5	2.8e-1
22	50	4.3e-4	2.6e+1	200	8.0e-5	5.2e+3
23	60	5.8e-5	2.6e+1	256	1.4e-4	3.6e+3
24	80	3.4e-5	1.5e+1	500	2.1e-4	5.2e+1
25	98	1.3e-4	3.8e-1	795	3.8e-5	3.3e+0
26	100	4.9e-5	7.2e+0	2000	1.3e-4	5.7e+2
27	120	3.1e-5	4.9e-2			
28	498	1.9e-4	4.8e-1			
29	2998	2.9e-5	3.3e-1			

$T_i^{(X)}$ is the numbers of cores used by class i in cell X. $p_i^{(X)}$ is the probability of class i in cell X. $\tau_i^{(X)}$ is the average core holding time in seconds for class i in cell X.

limit our study to classes in which jobs fall with reasonably high probability, also looking at the number of requested cores, so as not to overlook classes requesting very large numbers of cores. Thus, for the two cells that we consider, that we call Cell A and Cell B, we consider the data presented in Table I where $T_i^{(X)}$ is the number of cores used by class i in cell $X, X \in \{A, B\}$, $p_i^{(X)}$ is the probability of class i in cell X, $\tau_i^{(X)}$ is the average core holding time for class i in cell X.

In the case of Cell A, we consider 29 classes, with numbers of requested cores from 1 to almost 3,000, and core holding times spanning four orders of magnitude. In the case of Cell B, we consider 26 classes, with numbers of requested cores from 1 to 2,000, and core holding times spanning five orders of magnitude. In both cases, the least frequent job classes appear with probabilities of the order of a few units in 10^{-5}, which calls for simulation runs observing several million jobs to obtain credible performance estimates.

It should be noted that the average workload generated by jobs of Cell A with a job arrival rate equal to 1 s^{-1} is $L^{(A)} = \sum_{i=1}^{29} p_i T_i \tau_i = 5.78$, with average and standard deviation of the requested number of cores $T^{(A)} = 1.89$, $\sigma_{T^{(A)}} = 18.13$ and average and standard deviation of the overall service time $S^{(A)} = 2.47$ s, $\sigma_{S^{(A)}} = 8.32$ s. The corresponding numbers for Cell B are $L^{(B)} = 412.87$, $T^{(B)} = 8.59$, $\sigma_{T^{(B)}} = 35.1$, $S^{(B)} = 8.61$ s, $\sigma_{S^{(B)}} = 198.23$ s. We observe that Cell A is much less loaded than Cell B, due to both lower average number of cores and lower overall average service time. In both cases standard deviations are significantly larger than averages, thus confirming the high variability present in real data. Notice that, standard deviations are computed assuming that the measured average service times for each job class derive from exponential distributions. If we assume that all jobs in a class have identical service times (i.e., a deterministic pdf in each class), we get $\sigma_{S^{(A)}} = 5.62$ s and $\sigma_{S^{(B)}} = 140.04$ s. However, we also see that standard deviations in Cell A are lower than in Cell B. This hints at a much less problematic workload in Cell A with respect to Cell B.

C. Structure of the paper

The rest of this paper is organized as follows. Section II discusses the impact of service time variability on MJQM performance in simple settings through analytical and simulation approaches, comparing results with traditional queuing system behaviors. Section III presents and discusses simulation results for system workloads based on Google Borg measurements. Section IV discusses the possibility of improving performance by aggregating service demands with the objective of reducing the variability in terms of core requests. Section V looks at what performance gains would be possible if the variance of job service times could be reduced. Section VI summarizes the key results of the paper. Section VII discusses related works and finally Section VIII concludes the paper.

II. HOW SERVICE TIME DISTRIBUTIONS DETERMINE PERFORMANCE

As a first example of the impact of service demand variability on data center performance, we present analytical results in a simple MJQM setting, showing that the stability region depends on the job service time distribution.

The MJQM (Multiserver Job Queuing Model) is a queue with N identical servers and one FIFO (First-In First-Out) waiting line. In this section, we consider a simple MJQM model where jobs belong to just 2 classes, according to the number of servers they need. Larger systems, i.e., system with a larger number of classes, do not lend themselves to exact analysis, only to approximations, that are inaccurate by a constant error term, so that results become asymptotically exact for loads approaching the limit of the stability region [7]. Jobs of class 1 require just one server and are hence termed small. Jobs of class 2 require all the available N servers and are hence termed big. Services last for random times whose average is $\tau_1 = \tau_s$ s for small jobs and $\tau_2 = \tau_b$ s for big jobs. We refer to these delays as service times or holding times. Jobs arrive according to a Poisson process with rate λ [s^{-1}]. With probability $p_1 = p_s$ an arriving job is small, and with probability $p_2 = p_b = 1 - p_s$ an arriving job is big. Hence, the total workload of the queue is $\rho = \lambda p_s \tau_s + N\lambda p_b \tau_b$. The queue is not work-conserving because of the HOL blocking induced by the combination of FIFO and requests for a variable number of cores. As a result, the queue stability condition is not just $\rho < N$ as it would be in the case of no HOL blocking, and as it is for $p_s = 1$, when the MJQM reduces to an M/M/N queue (instead, when $p_s = 0$ the MJQM reduces to a M/M/1 queue, and the stability condition is $\rho < 1$). It is worth observing that whenever the MJQM is stable, ρ corresponds to the system

utilization, i.e., to the average number of busy servers. Since HOL blocking in some conditions prevents the full utilization of servers, the system stability is limited to values of utilization lower than N.

In [4], we derived the stability condition for this type of MJQM through the analysis of the behavior of a saturated system, i.e., of a system in which jobs are always present in the waiting line and occupy as many servers as allowed by HOL blocking, under the assumption of exponential service times for small jobs, also showing the invariance of the stability condition to the distribution of service times of big jobs. The analysis of a saturated MJQM is sufficient to compute several interesting performance metrics, such as the maximum throughput of each job class and of the entire queue, from which one can derive the system stability condition [6], [16]. The analysis in [4] is based on the study of the time interval T_{bb} between two consecutive completions of services of a big job in saturation. T_{bb} represents a renewal cycle for the stochastic process underlying the MJQM. Each renewal cycle comprises a geometrically distributed number of services of small jobs and one service of a big job.

Given $E[T_{bb}]$ in seconds, the throughput of big jobs (in big jobs per second) can be expressed as $X_b = \frac{1}{E[T_{bb}]}$, and the throughput of small jobs (in small jobs per second) as $X_s = \frac{p_s}{p_b} X_b$. The stability condition for the non-saturated MJQM is $\lambda p_b < X_b$. Hence, $\lambda p_b E[T_{bb}] < 1$, so that the stability region of the MJQM is $\lambda \in [0, \lambda_{\max})$, with

$$\lambda_{\max} = \frac{1}{p_b \; E[T_{bb}]} \,, \tag{1}$$

where $\lambda_{\max}$, measured in jobs per unit of time, represents the maximum workload intensity that can be served by the system.

In [4], we proved that the results of the cycle analysis are insensitive to the distribution of the service times of big jobs, depending only on τ_b. In this paper, we consider the cases of exponentially distributed and deterministic service times for small jobs and show that the MJQM stability condition does not only depend on the average service time of small jobs.

A. Exponential Service Times for Small Jobs

The exponential case was discussed in [4], where the average cycle time was obtained in [4, Eq. (3)] and can be expressed with a closed-form in terms of Euler's incomplete Beta function.

B. Deterministic Service Times for Small Jobs

If service times of small jobs are deterministic, hence identical for all small jobs, at the end of a service of a big job up to N small jobs enter service and they terminate τ_s seconds later, all at the same time. At this point, if more small jobs are in the waiting line before the next big job, a second batch of up to N small jobs enters service, and so on.

This means that, under the condition of k small jobs between two consecutive big jobs, the average cycle time duration can be expressed as:

$$E[T_{bb}|k] = \tau_b + \left\lceil \frac{k}{N} \right\rceil \tau_s \,. \tag{2}$$

Considering that the probability of k small jobs between two consecutive big jobs is $p_s^k \; p_b$ we obtain:

$$\begin{aligned} E[T_{bb}] &= \tau_b + \sum_{k=0}^{\infty} \left\lceil \frac{k}{N} \right\rceil \; \tau_s \; p_s^k \; p_b \\ &= \tau_b + \tau_s \; p_b \left[\sum_{i=1}^{\infty} \sum_{k=(i-1)N+1}^{iN} i \; p_s^k \right] \\ &= \tau_b + \tau_s \; p_b \left[\sum_{i=1}^{\infty} i \; \frac{p_s^{(i-1)N+1} - p_s^{iN+1}}{1 - p_s} \right] \\ &= \tau_b + \tau_s \; p_s \sum_{i=0}^{\infty} p_s^{iN} \,, \end{aligned}$$

so that finally

$$E[T_{bb}] = \tau_b + \frac{\tau_s p_s}{1 - p_s^N} \,. \tag{3}$$

and the stability region results from (1). It is worth noting that this is the first exact characterization of the stability region in a MJQM with no exponential service time assumptions.

C. Comparison of Stability Regions

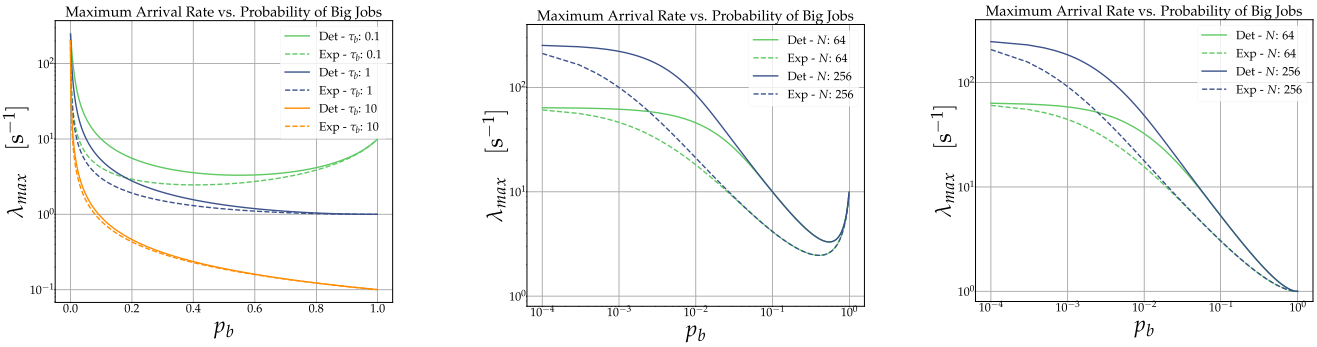
In Fig. 2(a), we plot the values of the maximum workload intensity, $\lambda_{\max}$ [s^{-1}], that define the limit of the stability region versus $p_b = 1 - p_s$ for different values of τ_b and for either exponential or deterministic service times of small jobs.

We see that for values of p_b close to 1, results for deterministic and exponential service times of small jobs tend to coincide. This is obvious, since for $p_b = 1$ we have $E[T_{bb}] = \tau_b$, independently of the service time of small jobs, and thus $\lambda_{\max} = 1/\tau_b$, independently of τ_s. Similarly, we see that curves come together for values of p_b close to 0. This is again obvious since, for $p_b = 0$, we have only small jobs, hence no HOL blocking and stability is driven by the number of available resources ($\lambda_{\max} = N$ [s^{-1}] since we set $\tau_s = 1$ s). Instead, for intermediate values of p_b we observe significant differences in the values of $\lambda_{\max}$, which result from the amount of resources wasted due to HOL blocking. Quite interesting (but not unexpected) is the observation that the stability region is always wider in the case of deterministic service times of small jobs, as well as the non-monotonic behavior of the stability region width for low values of τ_b.

Fig. 2(b) and (c) report the upper limits of the stability region of the MJQM in log-log plots. Observe that, for $p_b \to 0$ and $p_b \to 1$, the system's stability is insensitive to the exponential or deterministic distribution of the small jobs' service time. This happens because, in these cases, the system becomes work-conserving and the stability region is therefore insensitive to moments higher than the first of the service times.

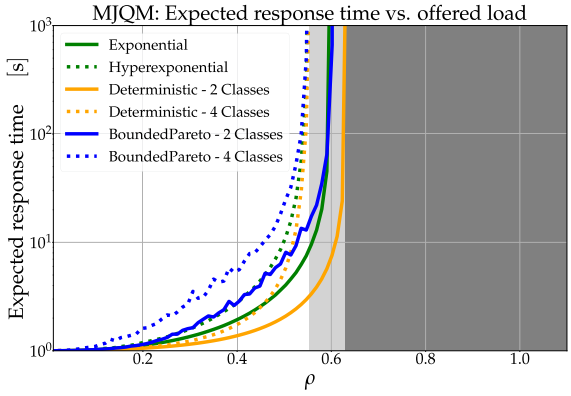
D. Two More MJQM

To convince the reader that the issue we just discussed is not a result of the peculiarities of the MJQM studied with the analytical model (that is constrained to just two job classes, one of which requires all cores), in Fig. 3(a) we report simulation results for the average overall response time versus the job arrival rate in a MJQM with 100 cores and either two or four classes of

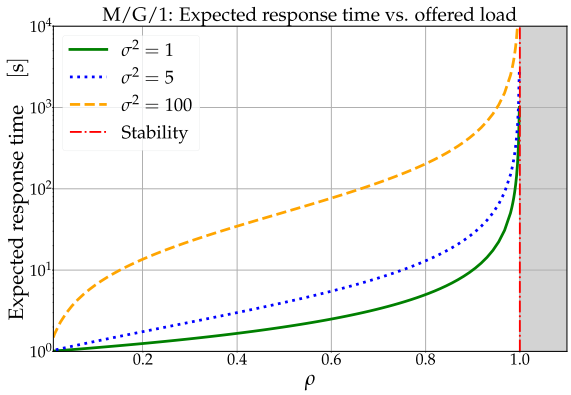


(a) N = 256 and τ_b = 0.1, 1, 10 s (b) N = 64, 256 and τ_b = 0.1 s (c) N = 64, 256 and τ_b = 1 s

Fig. 2. Maximum values of the arrival rate for stability versus average probability of big jobs p_b in MJQM with either $N = 256$ or $N = 64$ and $\tau_s = 1$ s, with either exponential or deterministic service times of small jobs



(a) MJQM expected response time



(b) M/G/1 expected response time

Fig. 3. MJQM and M/G/1 expected response time versus the fraction of used resources ρ, with average service time 1, for different values of service time variance

jobs. In this case, the largest number of requested cores (50) is half the total number of cores.

The service time distribution within each job class is assumed to be of three different types: i) deterministic, ii) negative exponential, iii) bounded Pareto (see, e.g., [17]) with $\alpha = 2$ and range from 0.50002 τ to 12000 τ. The number of cores allocated to classes, the class probabilities, and the average

TABLE II
PARAMETERS OF THE MJQM CONSIDERED IN FIG. 3(A)

i	$T_i^{(4)}$	$p_i^{(4)}$	$\tau_i^{(4)}$ [s]	$T_i^{(2)}$	$p_i^{(2)}$	$\tau_i^{(2)}$ [s]
1	1	0.6999	0.28575511	1	0.8	1
2	1	0.0001	3000	50	0.2	1
3	1	0.1	3			
4	50	0.2	1			

class service times are reported in Table II. In the case of exponential service times, the overall service time distribution is a hyperexponential in the case of four classes (as we explained in the introduction) and an exponential in the case of just two classes (this happens because the average service time is the same for both classes). The parameters of the MJQM in the Exponential (E) and Hyperexponential (H) cases produce the following averages and standard deviations: $L^{(E)} = 10.8$, $T^{(E)} = 10.8, \sigma_{T^{(E)}} = 19.6, S^{(E)} = 1.0$ s, $\sigma_{S^{(E)}} = 1.0$ s for the Exponential case and $\sigma_{S^{(H)}} = 42.44$ s for the Hyperexponential case (while $L^{(H)}$, $T^{(H)}$, $\sigma_{T^{(H)}}$, and $S^{(H)}$ remain like in case (E)). In particular, $S^{(E)} = S^{(H)}$, i.e., the average service time is the same in the two cases.

We wish to observe that case E is equivalent to case H, except for the fact that by giving all classes the same average service time $\tau_i = 1$ s, the resulting overall service time distribution is exponential in case E, while the different average service times of the four classes of case H make the overall service time distribution hyperexponential with fairly high variance. We can recognize in both cases a class of jobs requesting 50 cores with a probability 0.2, and either three or one classes requesting one core with an overall average service time equal to 1. The parameters of the MJQM in the deterministic case with two and four classes (D2 and D4) and bounded Pareto with two and four classes (P2 and P4) produce the following overall standard deviations of service times: $\sigma_{S^{(D2)}} = 0.0$ s, $\sigma_{S^{(D4)}} = 30.0$ s, $\sigma_{S^{(P2)}} = 2.01$ s, $\sigma_{S^{(P4)}} = 67.41$ s. It is very important to stress that the standard deviation of the overall service time is strongly influenced by the number of classes, as well as number of cores and service time pdf within classes.

From Fig. 3(a), we can see that both the average response time and the stability region are influenced by the higher moments of the overall service time distribution, which is the same effect we

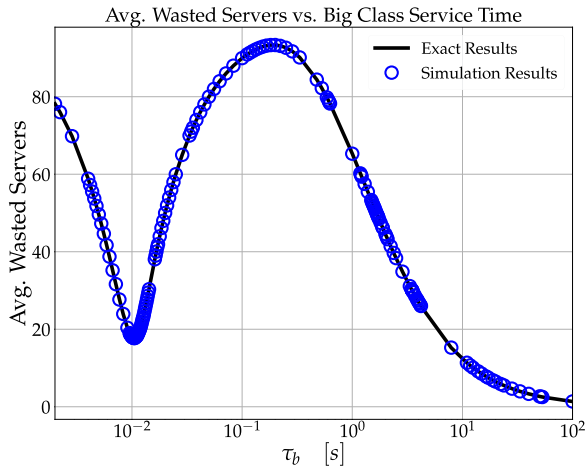


Fig. 4. Average number of servers wasted because of HOL blocking in a saturated MJQM-FIFO with small and big jobs versus average service time of big jobs. $N = 200$, small jobs request 1 server, big jobs request 100 servers, $\tau_s = 1$, $p_s = 0.5$. Comparison of simulation predictions against the exact analytical results of [3], [4]. The y-axis scale is linear.

observed in analytical results. In particular, we see that a lower standard deviation of the overall service time yields a lower average response time as well as a wider stability region (again like in the analytical case). The non-work-conserving nature of the MJQM is evident due to the observation that the maximum fraction of usable working power is considerably less than 1. Quite remarkable is the fact that the stability condition appears to be influenced more by the number of classes than by the service time pdf within classes. This is the result of what we already observed, i.e., of the strong dependence of the standard deviation of the overall service time on the number of classes.

E. The M/G/1 Case

Finally, in order to emphasize the difference in the impact of higher moments of the service time on the performance of different queuing systems, in Fig. 3(b), we report the average response time for an M/G/1 queue as a function of the load ρ (that corresponds to the arrival rate, since we assume an average service time equal to 1 s). The service time variance takes values 1, 5, and 100. The M/G/1 queuing system, a widely adopted model, features a single server where jobs require random service times. Jobs arrive at the system following an independent Poisson process and are served in FIFO order. The Pollaczek-Khinchine formula enables us to express the expected response time as a function of job arrival rate, mean, and variance of service time. Fig. 3(b) shows that the sensitivity to the variance of the service time regards only the expected response time, while the stability region can be observed to be insensitive to the second moment of the distribution, always coinciding with the interval [0,1).

Remark 1 (Peculiarity of the MJQM stability region): With this remark, we highlight the distinctive characteristics of the stability region of the MJQM studied in this paper.

The stability region of the MJQM, as well as several performance metrics, are influenced by higher moments of the distributions of job service times and of the number of allocated cores.

This stems from the non work-conserving characteristic of the MJQM, due to HOL blocking.

Remark 1 is extremely relevant for the MJQM management since it provides indications on the factors impacting performance. If service times are hardly modifiable, the numbers of allocated cores can be carefully chosen.

III. Variability in Service Demands: Which Consequences in Real Systems?

In this section, we present and comment on simulation results obtained by driving the MJQM with the data measured in a Google Borg cloud data center. The simulation experiments were built with a custom MJQM simulation package written in C++ that can implement a number of different MJQM workloads and schedulers. The simulation package will be made available to researchers at the time of publication of this work.

The simulation software utilizes the `mt19937_64` instantiation of the Mersenne-Twister pseudo-random number generator (RNG). The latter has a period defined by Mersenne prime $2^{19936 \cdot 64} - 1$, providing an exceptionally long sequence of pseudo-random numbers before repetition occurs. During each simulation run, the RNG samples a number of values equal to the total number of events. Subsequent runs are free from correlation risks, as they continue sampling from the same RNG sequence without exhausting the period. This ensures no overlap or repetition across runs due to the RNG's extensive cycle length. The RNG is then reinitialized for each different experiment (i.e., set of parameters). Additionally, following the methodology in [18], we further validated our approach by creating two separate streams for the stochastic components of our model, i.e., job arrivals and departures for each class. To achieve this, we used the same RNG mentioned earlier, configuring two distinct generator objects. The second generator skips a suitable number of draws from the RNG to avoid reusing the same random numbers. This approach produced comparable results, in the sense that confidence intervals significantly overlap. We run simulation experiments on a cluster node composed of 20 core Intel(R) Xeon(R) Gold 6148 CPU @ 2.40 GHz, 200 GB of ECC RAM. Storage is on a 30TB NAS, and everything is hosted on a Nutanix hyperconvergent architecture. With exponential service times, each independent run requires approximately 3 minutes, and for every set of parameters, we obtained 95% confidence intervals for the mean system population of the order of a few percent of the reported values, on average (higher when close to instability and when the estimated value is very small).

The plot in Fig. 4 presents sample results for the validation of our simulation package, comparing analytical and simulation results for a saturated MJQM-FIFO [3], [4]. The close alignment between simulation results and analytical predictions confirms the accuracy of our simulation package, validating it against a reliable baseline.

We simulated the dynamics of Cells A and B, using the parameters in Table I, assuming that the service time distribution for jobs within a class can be of four types: deterministic, uniform with support equal to the mean (i.e., with support from half the mean to 1.5 times the mean), negative exponential, bounded Pareto, in the last case using $\alpha = 2$, the minimum equal to about half the mean, and the maximum set to 12 thousand times the

TABLE III
STANDARD DEVIATION OF OVERALL SERVICE TIME WITH ORIGINAL DEMANDS AND DIFFERENT PDF OF SERVICE TIMES WITHIN CLASSES (BUT THE SAME FOR ALL CLASSES)

Distribution	$\sigma_S^{(B)}$	$\sigma_S^{(A)}$
Deterministic	140.04	5.62
Uniform (support equal to $[0.5\tau, 1.5\tau]$)	145.78	5.89
Exponential	198.23	8.32
Bounded Pareto ($\alpha = 2$, from 0.50002 τ to 12000 τ)	314.96	13.56

Average workload 412.87; average requested number of cores 8.59; standard deviation of requested number of cores 35.1; average service time 8.61 s. τ is the class average.

mean. The average service time within each class, and overall, remain the same in all cases, so that the system workload does not change. The resulting overall standard deviations in the different cases are reported in Table III. It is interesting to observe that the overall standard deviation is quite high, even with deterministic service times. This due to the presence of several job classes with quite different average service times and probabilities. We let the available number of cores be 3072 in Cell A and 2048 in Cell B, so as to accommodate jobs with the highest demand in terms of the number of cores.

In experiments featuring bounded Pareto distributed service times, we carried out 100 million events across 60 independent runs. For all other parameter sets, we conducted 40 independent replications, each comprising 30 million events.

Let us start by looking at the amber curves in Figs. 5 and 9, that refer to the unmodified data of Cells A and B, assuming an exponential service time distribution for jobs within each class, hence a hyperexponential overall service time distribution. In doing this, we must consider that for both cells an upper bound of the stability region is given by value λ_{ideal} that is the maximum traffic intensity in the *ideal* case of no resource waste due to HOL blocking. This can be computed as the ratio between the total service capacity, i.e., the number of available cores, and the workload for $\lambda = 1\,[\text{s}^{-1}]$. We thus obtain $\lambda_{ideal}^{(A)} = 3072/5.78 = 531.5\,[\text{s}^{-1}]$ and $\lambda_{ideal}^{(B)} = 2048/412.87 = 4.96\,[\text{s}^{-1}]$. The amber curves in Fig. 5 report the average response time computed over all jobs, as well as for the smallest and the biggest classes of jobs (the smallest class requests 1 core in both cells, while the biggest class requests 2998 cores in Cell A and 2000 cores in Cell B, as can be seen in Table I) in terms of the requested number of cores, as functions of the job arrival rate λ (simulations produce results for all classes, but in order not to clutter figures with too many curves, in Fig. 5 we report results only for the smallest and the biggest classes) The results for the average waiting time of all classes in the two cells (29 in Cell A and 26 in Cell B) are reported in Fig. 6. We see that all classes except the biggest one have very similar performance, while the biggest class suffers significantly longer waiting times, as expected, due to a greater difficulty in entering service. The overall average response and waiting times with different service time distributions within classes for the two cells are presented in Figs. 7 and 8. We can see that performance worsens with increasing overall service time variance, as expected, and that the system stability region shrinks, coherently with what we previously observed in simpler settings.

From Fig. 5 we can also see that the stability region in both cells only reaches up to about 30% (29.4 % for cell A and 31% for Cell B) of the λ_{ideal} values computed above. This is a very disappointing performance, indicating that about 70% of the installed computing capacity cannot be exploited by the cell workload due to the FIFO policy and the induced HOL blocking, which is aggravated by the extreme variability in service demands. In Fig. 9(a), we report for Cell B the average waiting time computed over all jobs (left plot) and the average fraction of resources that are allocated to jobs and of those that are actually used. The latter two quantities coincide in the case of the amber curves since the allocation of cores to jobs is identical to the job request. Also in this case, we see that no more than about 30% of the available computing capacity (and of the number of cores) can be accessed. The next section will discuss possibilities for the improvement of this quite disappointing performance.

IV. AGGREGATION OF RESOURCE DEMANDS

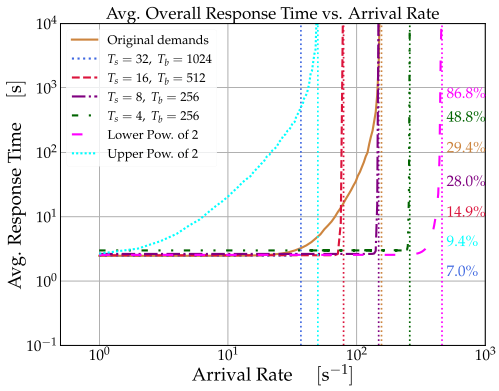
The variability in the workload generated by jobs refers to the job duration and to the number of cores requested by the job. While the duration of jobs is not known at the time when the scheduler must operate and asking the job owner for an estimate (as some popular job schedulers do) can be problematic, since this leads to overestimates to avoid job killing in case of overruns and consequent scheduling errors, the number of requested cores is known. We thus propose to act on core allocation in order to reduce the variability of the job service demand in this dimension.

A. The Aggregation Policy

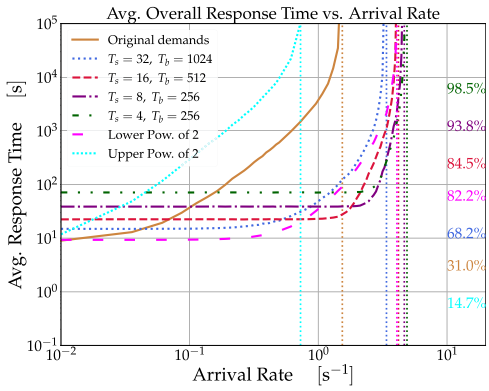
In order to reduce the variability in the job service demand, we propose to group job classes (remember that all jobs in a class request the same number of cores) into a small number of groups, allocating the same number of cores to all jobs in all classes belonging to one group. By doing so, all jobs in all classes of the same group are allocated the same number of cores. This implies that some jobs are allocated more cores than requested, and other jobs receive fewer than requested. In the former case, the core overallocation can increase the perceived system workload, since the scheduler is likely to waste resources, because a job often is not able to exploit a degree of parallelism higher than that corresponding to the number of requested cores (this assumption is analogous to the one made in [19] for inelastic jobs and coherent with the parallelizability limit defined in [20]). In the latter case, multiple tasks must share the same core, leading to longer service times. We assume that service times increase by a factor N_R/N_A where N_R is the number of requested cores and N_A is the number of allocated cores. We can formally define this transformation of the service times with the following equation:

$$\tau_{new} = \frac{N_R}{N_A}\,\tau_{old}. \tag{4}$$

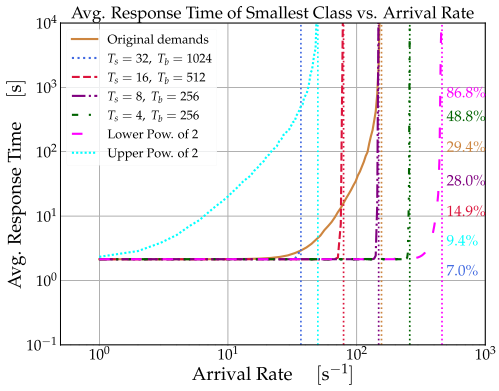
Notice that these assumptions create a different and, in some sense, more challenging setting than that studied in [21], where the authors assume that when a job receives more cores than it requests, its service time becomes shorter. It is also important to note that this aggregation methodology results in varying behaviours regarding the average and standard deviation of service times, depending on the original demands. For cell



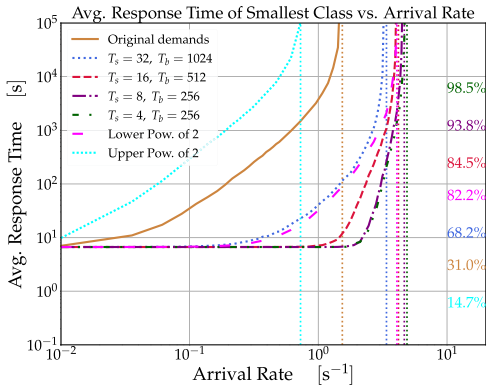
(a) Cell A, $N = 3072$



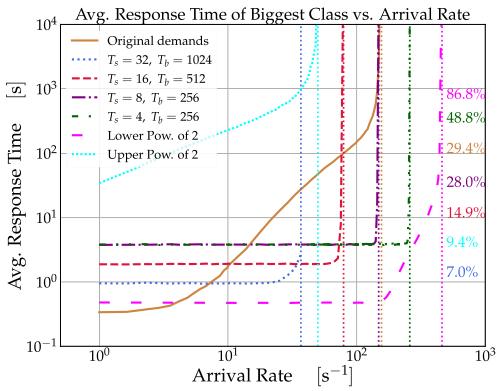
(b) Cell B, $N = 2048$



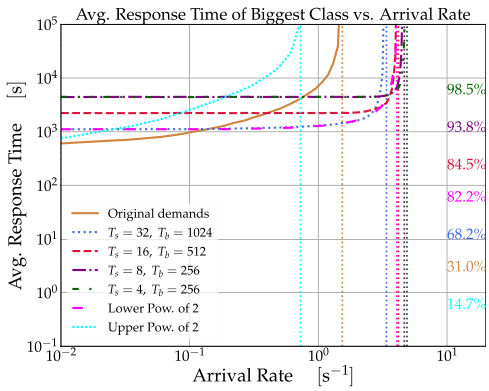
(c) Cell A, $N = 3072$



(d) Cell B, $N = 2048$



(e) Cell A, $N = 3072$



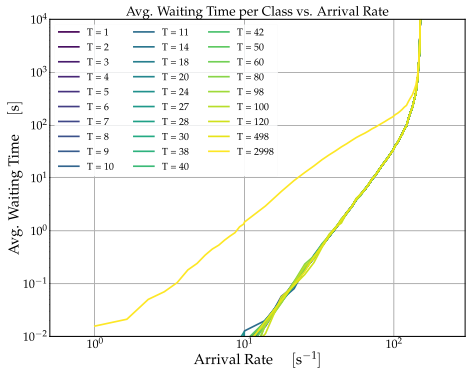
(f) Cell B, $N = 2048$

Fig. 5. Average response time vs. job arrival rate for cells A and B; (a) and (b) show average overall jobs, (c) and (d) average over jobs of smallest class; (e) and (f) average over jobs of biggest class.

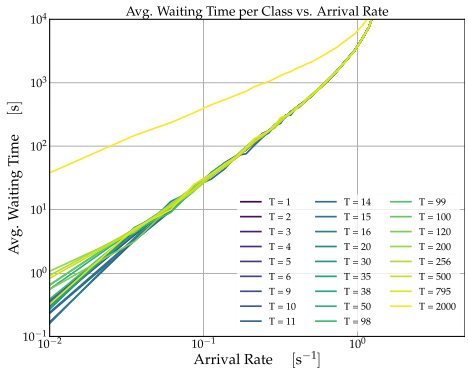
A, all tested scenarios tend to maintain a similar average and standard deviation compared to the original demands or show a slight increase. In contrast, for cell B, the results tend to differ significantly. This is particularly interesting because, as seen in Fig. 5, stability conditions and performance indices are consequently influenced by more than just the first and second moments.

B. *Performance Evaluation*

The curves in Figs. 5 and 9 report results for six different cases of job class grouping. In the first four cases, we use only two groups for job classes, aggregating in one group all classes but the last one, which is the biggest class (the class whose jobs request the largest number of cores). Using T_b and T_s to indicate

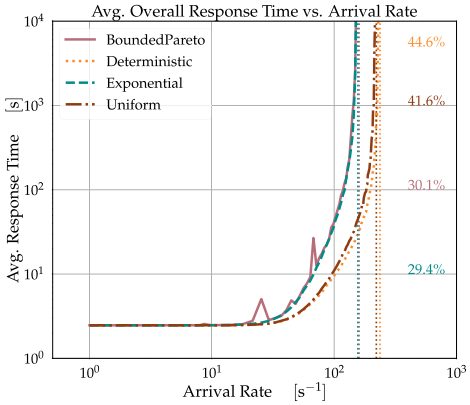


(a) Cell A, $N = 3072$

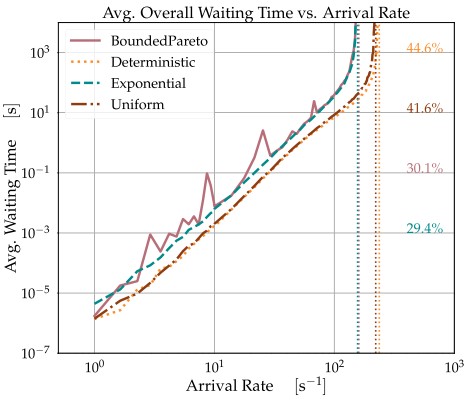


(b) Cell B, $N = 2048$

Fig. 6. Average waiting time per each class vs. job arrival rate for cells A and B

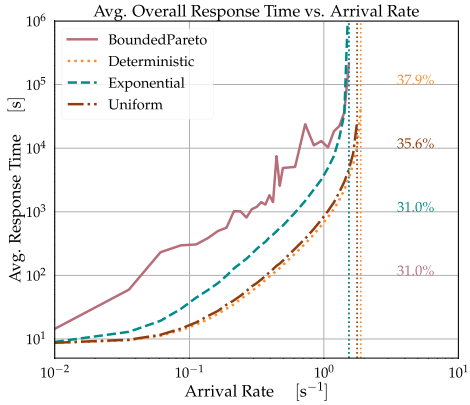


(a) Average Overall Response Time Vs. Arrival Rate

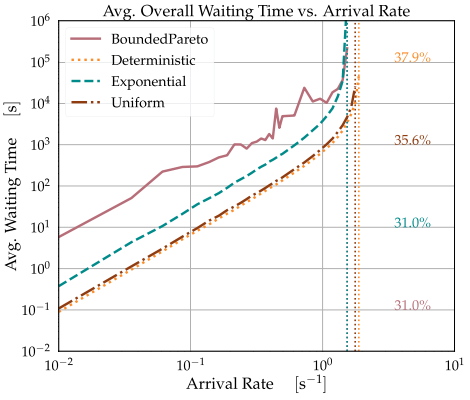


(b) Average Overall Waiting Time Vs. Arrival Rate

Fig. 7. Overall Metrics (Original Demands) Cell A



(a) Average Overall Response Time Vs. Arrival Rate



(b) Average Overall Waiting Time Vs. Arrival Rate

Fig. 8. Overall Metrics (Original Demands) Cell B

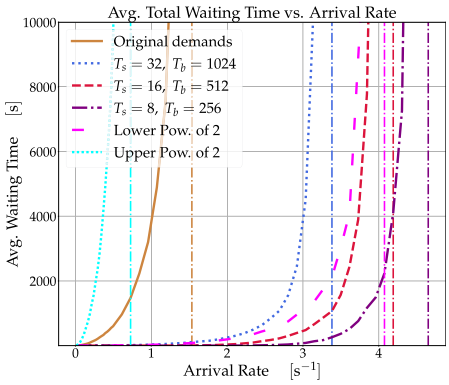
the number of cores allocated respectively to the biggest class and to all others, these four cases are:

- Allocation 1: $T_b = 1024$, $T_s = 32$ (blue curves)
- Allocation 2: $T_b = 512$, $T_s = 16$ (red curves)
- Allocation 3: $T_b = 256$, $T_s = 8$ (purple curves)
- Allocation 4: $T_b = 256$, $T_s = 4$ (green curves)

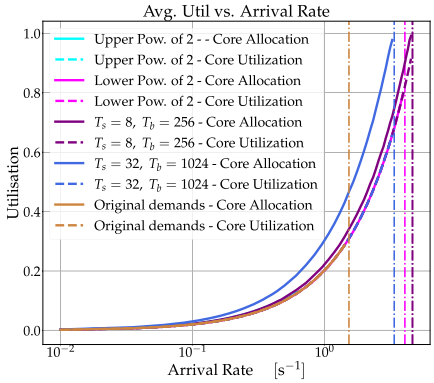
We also consider two more groupings, where all job classes in a group are allocated the same number of cores, equal to a power of two. The power of two is either the one just below the class request, or the one just above. We call these two groupings:

- Allocation 5: upper power of 2 (light blue dotted curves)
- Allocation 6: lower power of 2 (pink dashed curves)

The only exception to the power of 2 rule is in the case of the biggest class in Cell A, which requires 2998 cores and should



(a) Cell B, waiting time. Both axis are in linear scale.



(b) Cores Allocation and Utilization. y-axis is in linear scale.

Fig. 9. Average waiting time, core allocation, and utilization for cell B.

be allocated 4096; however, the cell only has $N = 3072$ cores, and thus it is not possible to allocate more than 3072.

Notice that, in each experimental scenario we maintain the same number of classes as specified by the original demands. When referring to grouping, we mean that each class is allocated a certain amount of cores (e.g., T_s).

The aggregation performed on the job classes brings the service time of each job to be adjusted using Eq. (4), if the job receives less cores than requested. We wish to repeat here that, for example, the allocation of 32 cores to a job requesting just 1 does not produce a variation of the job execution time and implies the waste of 31 of the 32 allocated cores. However, the reduced granularity in the core allocations has the very positive effect of reducing the impact of HOL blocking.

While the optimization of the class groupings is an obviously interesting issue, on which we will report in future works, here we only experiment with simple groupings, to show that this can be an important element for the improvements of the MJQM performance.

The reasons that motivated our choice of groupings 1 to 4 are the following: i) the biggest class is given a larger number of cores in order to avoid extending too much the execution time of jobs in this class; ii) all other classes are given the same small number of cores to reduce as much as possible the service demand variability; iii) the number of cores allocated to the biggest class is set to be a multiple of the number of cores allocated to all other classes in order to be able to fully utilize the cores freed by the end of service of a big class job, using jobs of any class. Instead, groupings 5 and 6 were considered to experiment with larger numbers of groups (13in Cell A and 12 in Cell B), while still maintaining a constant factor between group sizes, in order to facilitate the utilization of cores upon completion of a job.

The results of job class grouping can be observed in the curves in Figs. 5 and 9. First of all, we can observe that the upper power of 2 grouping (Allocation 5) yields poor performance, since many cores are wasted, and this waste is not compensated by the ease of core utilization at the end of a service. On the contrary, the lower power of 2 grouping (Allocation 6) yields extremely good performance, since only few cores are wasted on average, and core utilization at the end of a service is made easier by the fact that jobs are allocated cores in multiples of 2.

Coming to the simpler class groupings that just use 2 groups, in some cases we see very good performance. In Cell A, with Allocation 1 we are limited to just 7% of the theoretical MJQM capacity, with a significant loss with respect to the amber curve that corresponds to no grouping of classes and reaches 29.4%. However, with Allocation 4 we reach 48.8%, which is much higher than the 29.4% achieved by the amber curve. Results are much more favorable in the case of Cell B, since now Allocation 1 achieves 68.2% of the theoretical capacity, and Allocation 4 obtains 98.5%. In addition, we see that, for a wide range of job arrival rates, the considered allocations yield much lower average overall response time with respect to the case of no grouping. For example, Allocation 1 performs better than the case of no grouping with job arrival rates higher than about 0.05. With job arrival rate equal to 1, Allocation 3 reduces of about two orders of magnitude the average response time with respect to the case of no grouping at an arrival rate equal to 1 job/s. Quite interesting is the fact that this gain is not obtained at the expense of some of the job classes. As we can see in Fig. 5, both the classes with the largest and the smallest request in terms of number of cores benefit from grouping. Similar benefits are achieved by other classes, but results are not reported for the sake of brevity.

The comparison between Allocation 6 (lower powers of 2) and Allocations 1-4 shows that the latter always perform worse in Cell A, while in Cell B they improve performance at medium-high loads, thanks to the result for smaller job classes.

We repeat once more that this excellent result is obtained in spite of the extra allocation of cores to some job classes.

We wish to conclude this discussion by reporting that additional simulations, with an allocation of 1 core to all job requests, that call for rescaling all service times except those of the smallest class, reach 100% of the theoretical capacity. This is obvious, since by allocating 1 core to all jobs we go back to a traditional M/M/N queue behavior, with no extra allocation of cores and no HOL blocking. However, this result is an indication on the one hand of the correct behavior of our simulator, and on the other of the drastic impact on performance of variability in job demands.

V. Reduction of the Service Time Variance

Although we are well aware that what we investigate in this section is hardly doable in practice, we now look at the performance of Cell B assuming we are able to reduce the

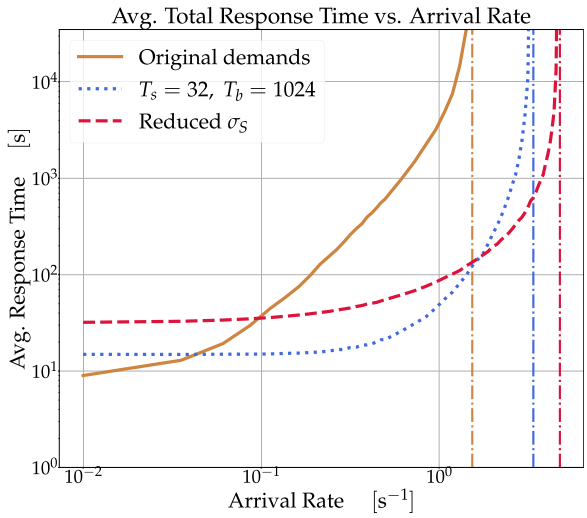


Fig. 10. Cell B, comparison among original data without grouping, job class grouping, and service time variance reduction.

variance of job service times. The reason for this investigation is to understand whether the variability in job service times and cores requests have similar impacts on the MJQM performance.

In Fig. 10, we plot the average response time in three cases: i) original Cell B data (amber curve); ii) job class grouping with core allocation $T_b = 1024$ and $T_s = 32$ (blue curve); iii) reduced service time variance (red curve). While the first two cases correspond to curves we have already included in Figs. 5 and 9, the third curve is obtained by letting the first 25 job classes share the same average service time, which is set equal to the overall average service time of the same 25 classes in the original data. This corresponds to forcing the first 25 job classes to share the same exponential service time distribution, thus reducing the overall service time variance while keeping the same workload as in the original Cell B data. It is quite interesting to observe that the effect of this reduction of the service time variance is not far from the one obtained with job class grouping, that reduces the variance in core allocation. This tells us that the service demand variability impacts performance on both service dimensions defined by core demands and service time demands.

VI. Major Takeaways

In this section, we summarize the major takeaways of the combination of analytical and simulation results that we have presented.

- *Variability in jobs' demands matters:* In the realm of work-conserving systems, the moments of the service demand higher than the first may have an impact on some performance measures such as the expected job response time, not on system stability. On the contrary, we have analytically shown that in the context of MJQM (that are not work-conserving), these moments play a pivotal role also in defining the system's stability region, i.e., the maximum workload intensity that can be served before reaching saturation. This phenomenon arises from the inefficient utilization of resources caused by the granularity in jobs' resource demands that causes HOL blocking and is aggravated by high service time variability. This effect on the one side has significant theoretical importance, and on the other heavily impacts the performance of real-world data center workloads. A careful investigation of the impact of moments of different orders is left for future work. Our preliminary results indicate an impact of moments of order two and higher, as shown also in [22], where the the intuition that higher variances of the service time distribution for small jobs would lead to more restrictive stability conditions is shown to be not always true with a counterexample.
- *Increase the service time to reduce the response time in moderate/heavy load conditions:* When a job is allocated more resources than it actually demands, it might not fully utilize them, resulting in a minimal reduction in its service time (in our simulations we assume that the service time does not change). On the contrary, if a job receives fewer resources than it requires, its service time is likely to be prolonged in direct proportion to the shortfall. Nevertheless, our findings reveal that allocating additional resources to smaller jobs has a minimal impact on overall system utilization when just two groups are used. In fact, they usually have short service times and the surplus resources they receive would be left unused by the HOL blocking. Conversely, decreasing resources allocated to larger jobs diminishes the likelihood of HOL blocking. The combined effects of these two strategies can substantially expand the system's stability region, consequently reducing the average response times under moderate to heavy load conditions.
- *System performance can be improved using only a few core allocations:* In order to increase the maximum reachable system throughput, it is sufficient to cluster jobs in a very limited number of class groups and allocate a fixed number of cores to all classes in a group. This is particularly evident in Fig. 9(a) where the gain in stability region is significantly greater for the aggregation $T_s = 8, T_b = 256$ compared to the *power of 2* aggregations. In our simulations based on the Google Borg traces, we used different settings for the aggregation of classes and showed that even for relatively low-intensity workloads, it is possible to reduce the expected response time both for large and small jobs thanks to the mitigation of HOL blocking effects achieved by grouping. This remains true until the point where resource waste becomes excessive due to overallocation, which is the case of the *upper power of two allocation*.

VII. Related Work

The analysis of the MJQM with FIFO scheduling holds a crucial role for the enhancement of contemporary data center performance evaluation [23], [24], [25].

As highlighted in [2], [26] the availability of results on the analysis of the FIFO-MJQM is limited. In [4], [6] the authors investigated the system, giving some valuable insight on this specific model. Notably, these works establish a stability condition for the MJQM with class-dependent service rates for the first time, considering systems with two classes of jobs under a saturation assumption. Following the same methodology

as in [4], we defined the stability region for the two class saturated MJQM with deterministically distributed service time in Section II-B. Apart from these investigations, researchers have primarily focused their attention on analyzing different scheduling policies, as exemplified in [12], [27], [28], [29], [30], on studying predecessors of this model [31], [32], [33] or on looking into this very same system but under various and strict constraints and assumptions, as in [34], [35], [36].

In an attempt to enhance both theoretical and practical advancements into the efficient management and optimization of the performance of modern computational systems, major companies such as Google [37], [38], Azure [39], [40], and Alibaba [41] have chosen to release comprehensive job-scheduling traces extracted from their cluster management systems. A significant contribution in this domain can be found in the work presented in [1], where the authors conducted detailed comparisons and analyzes of traces provided by Google through its Borg [38] cluster management system. This research highlighted the intricate dynamics of these systems, particularly emphasizing the remarkable variability in the distribution of requested CPUs within the Borg framework.

Other studies, such as [42] and [43], have provided similar insights into this challenge by analyzing the priority mechanisms of the Google Cloud public dataset. The characterization of machines and jobs is then further explored in [44]. There, the authors observe a prevalence of single-task jobs, with the majority containing less than 100 tasks and a very small fraction having up to 2000 tasks. Furthermore, in terms of job categorization, short-duration jobs emerge as both the most frequent and resource-efficient, contrasting with medium jobs, which are frequent and exhibit a memory-intensive nature, and the least common but very resource-demanding, long-duration jobs are identified as CPU-intensive. Building upon this research direction, [14] also offers insights into job resource usage patterns. The study observes again the highly diverse resource requests per task, and the authors reveal that the overall memory usage is reported to be only 53% of the memory allocation, while CPU usage corresponds to a mere 40% of the allocated CPU resources, emphasizing one more time the high diversity and often inefficient utilization of resources in modern computation systems.

With the purpose of understanding and addressing the challenges posed by the above-mentioned counter-intuitive features, we propose what, to the best of our knowledge, is the first study focused on clarifying how and how much service demand variability, in terms of resource allocation and service duration, impacts data centers performance. We employ a combination of simple analytical models and detailed simulations based on real-world data to delve into this complex domain.

VIII. Conclusion

This paper showed that the stability region of multiserver job queuing models depends not only on the average service demand of jobs, but also on higher moments of both the number of requested cores, and service duration. This was proved with analytical models in simple cases for which analysis is possible, and observed with simulation experiments in controlled conditions, as well as with simulations that use data extracted from publicly available measurements of Google Borg data centers.

The results we presented are relevant from the theoretical viewpoint, since the stability region in queuing systems quite often only depends on averages, and originates from the non work-conserving nature of the multiserver job queuing model. Our results are also relevant from a practical viewpoint, since they unveil the possibility of disappointingly low resource utilization levels in data centers.

However, the insight generated by our analysis paves the way to the design of approaches to improve resource utilization. They suggest that the aggregation of jobs requesting different numbers of cores in few groups where all jobs are allocated the same number of cores, can be quite beneficial in terms of performance, succeeding in reducing the average response time of jobs, even if their execution time is increased whenever the number of allocated cores is less than what requested. It is worth noting that the fact that only few class groups are sufficient to significantly improve the cluster's performance corresponds also to a simplification of the implementation of this strategy.

As future work, we leave the problem of determining the optimal job class grouping for the minimization of the expected response time or the maximization of some reward function. This function could encompass fees paid by jobs for the services rendered or energy efficiency.

References

[1] M. Tirmazi et al., "Borg: The next generation," in *Proc. Eur. Conf. Comput. Syst.*, 2020, pp. 1–14.

[2] M. Harchol-Balter, "The multiserver job queueing model," *Queueing Syst., Theory Appl.*, vol. 100, pp. 201–203, 2022.

[3] I. Grosof, M. Harchol-Balter, and A. Scheller-Wolf, "Stability for two-class multiserver-job systems," 2020, *arXiv:2010.00631*.

[4] D. Olliaro, M. Ajmone Marsan, S. Balsamo, and A. Marin, "The saturated multiserver job queuing model with two classes of jobs: Exact and approximate results," *Perf. Eval.*, vol. 162, 2023, Art. no. 102370.

[5] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge, U.K.: Cambridge Univ. Press, 2013.

[6] I. Grosof, M. Harchol-Balter, and A. Scheller-Wolf, "New stability results for multiserver-job models via product-form saturated systems," *SIGMETRICS Perform. Eval. Rev.*, vol. 51, pp. 6–8, 2023.

[7] I. Grosof, Y. Hong, M. Harchol-Balter, and A. Scheller-Wolf, "The reset and marc techniques, with application to multiserver-job analysis," *Perf. Eval.*, vol. 162, 2023, Art. no. 102378.

[8] The Apache Software Foundation, "Faircallqueue - apache hadoop," 2024. [Online]. Available: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/FairCallQueue.html

[9] A. Mu'alem and D. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 6, pp. 529–543, Jun. 2001.

[10] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "Characterization of backfilling strategies for parallel job scheduling," in *Proc. Int. Conf. Parallel Process. Workshop*, 2002, pp. 514–519.

[11] I. Grosof and M. Harchol-Balter, "Serverfilling: A better approach to packing multiserver jobs," in *Proc. ACM Workshop Adv. Tools Program. Lang. Platforms Implementing Evaluating Algorithms Distrib. Syst.*, 2023, pp. 1–5.

[12] I. Grosof, Z. Scully, M. Harchol-Balter, and A. Scheller-Wolf, "Optimal scheduling in the multiserver-job model under heavy traffic," in *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 6, p. 1–32, 2022.

[13] G. Amvrosiadis, J. W. Park, G. R. Ganger, G. A. Gibson, E. Baseman, and N. DeBardeleben, "On the diversity of cluster workloads and its impact on research results," in *Proc. USENIX Annu. Techn. Conf.*, 2018, pp. 533–546.

[14] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. ACM Symp. Cloud Comput.*, 2012, pp. 674–687.

[15] M. Yildiz and A. Baiocchi, "Data-driven workload generation based on Google data center measurements," in *Proc. IEEE 25th Int. Conf. High Perform. Switching Routing*, 2024, pp. 143–148.

[16] F. Baccelli and S. Foss, "On the saturation rule for the stability of queues," *J. Appl. Probab.*, vol. 32, pp. 494–507, 1995.
[17] L. M. Leemis and S. K. Park, *Discrete-event Simulation: A First Course*. Pearson Prentice Hall, 2006.
[18] I. B. Aban, M. M. Meerschaert, and A. K. Panorska, "Parameter estimation for the truncated pareto distribution," *J. Amer. Stat. Assoc.*, vol. 101, no. 473, pp. 270–277, 2006.
[19] B. Berg, M. Harchol-Balter, B. Moseley, W. Wang, and J. Whitehouse, "Optimal resource allocation for elastic and inelastic jobs," in *Proc. ACM Symp. Parallelism Algorithms Archit.*, 2020, pp. 75–87.
[20] B. Berg, B. Moseley, W. Wang, and M. Harchol-Balter, "Asymptotically optimal scheduling of multiple parallelizable job classes," 2024, *arXiv:2404.00346*.
[21] B. Berg, J. Whitehouse, B. Moseley, W. Wang, and M. Harchol-Balter, "The case for phase-aware scheduling of parallelizable jobs," *Perf. Eval.*, vol. 153, 2022, Art. no. 102246.
[22] A. Anggraito, D. Olliaro, M. A. Marsan, and A. Marin, "Stability of the multiserver job queuing model with infinite resources," in *Proc. Anal. Stochastic Modelling Techn. Appl.*, 2024, Art. no. 148.
[23] L. Sliwko, "A taxonomy of schedulers – operating systems, clusters and big data frameworks," *Glob. J. Comput. Sci. Technol.*, vol. 19, pp. 25–39, 2019.
[24] S. H. H. Madni, M. S. Abd Latiff, M. Abdullahi, S. M. Abdulhamid, and M. J. Usman, "Performance comparison of heuristic algorithms for task scheduling in IaaS cloud computing environment," *PLoS One*, vol. 12, pp. 1–26, 2017.
[25] D. Feitelson, L. Rudolph, and U. Schwiegelshohn, "Parallel job scheduling - a status report," in *Proc. Int. Workshop Job Scheduling Strategies Parallel Process.*, 2004, pp. 1–16.
[26] M. Harchol-Balter, "Open problems in queueing theory inspired by datacenter computing," *Queueing Syst., Theory Appl.*, vol. 97, pp. 3–37, 2021.
[27] B. Speitkamp and M. Bichler, "A mathematical programming approach for server consolidation problems in virtualized data centers," *IEEE Trans. Serv. Comput.*, vol. 3, no. 4, pp. 266–278, Fourth Quarter, 2010.
[28] K. Psychas and J. Ghaderi, "Randomized algorithms for scheduling multi-resource jobs in the cloud," *IEEE/ACM Trans. Netw.*, vol. 26, no. 5, pp. 2202–2215, Oct. 2018.
[29] L. Georgiadis, M. J. Neely, and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," *Found. Trends Netw.*, vol. 1, pp. 1–144, 2006.
[30] R. Vaze, "Scheduling multi-server jobs is not easy," 2024, *arXiv:2404.05271*.
[31] E. Arthurs and J. S. Kaufman, "Sizing a message store subject to blocking criteria," in *Proc. Perf. Comput. Syst.*, 1979, pp. 547–564.
[32] N. M. van Dijk, "Blocking of finite source inputs which require simultaneous servers with general think and holding times," *Oper. Res. Lett.*, vol. 8, pp. 45–52, 1989.
[33] W. Whitt, "Blocking when service is required from several facilities simultaneously," *AT&T Tech. J.*, vol. 64, no. 8, pp. 1807–1856, 1985.
[34] L. Afanaseva, E. Bashtova, and S. Grishunina, "Stability analysis of a multi-server model with simultaneous service and a regenerative input flow," *Methodol. Comput. Appl. Probab.*, vol. 22, pp. 1439–1455, 2020.
[35] A. Rumyantsev and E. Morozov, "Stability criterion of a multiserver model with simultaneous service," *Ann. Oper. Res.*, vol. 252, pp. 29–39, 2017.
[36] E. Morozov and A. S. Rumyantsev, "Stability analysis of a MAP/M/s cluster model by matrix-analytic method," in *Proc. Eur. Workshop Comput. Perform. Eng.*, 2016, pp. 63–76.
[37] J. Wilkes, "More Google cluster data," 2011. [Online]. Available: https://research.google/blog/more-google-cluster-data/
[38] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proc. Eur. Conf. Comput. Syst.*, 2015, pp. 1–17.
[39] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proc. Symp. Operating Syst. Princ.*, 2017, pp. 153–167.
[40] O. Hadary et al., "Protean: VM allocation service at scale," in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, 2020, pp. 845–861.
[41] Developer Alibaba, "Alibaba cluster data: Using 270 GB of open source data to understand Alibaba data centers," 2019. [Online]. Available: https://www.alibabacloud.com/blog/594340
[42] P. Minet, E. Renault, I. Khoufi, and S. Boumerdassi, "Analyzing traces from a Google data center," in *Proc. IEEE Int. Wireless Commun. Mobile Comput. Conf.*, 2018, pp. 1167–1172.
[43] D. Lasantha and B. Ray, "Priority based modeling and comparative study of Google cloud resources between 2011 and 2019," in *Proc. IEEE 19th Int. Conf. Trust Secur. Privacy Comput. Commun.*, 2020, pp. 1310–1317.
[44] Z. Liu and S. Cho, "Characterizing machines and workloads on a Google cluster," in *Proc. IEEE 41st Int. Conf. Parallel Process. Workshops*, 2012, pp. 397–403.

Diletta Olliaro is currently working toward the PhD degree in computer science with the Ca' Foscari University of Venice. She served as local chair for the 16th Joint Conference ACM SIGMETRICS / IFIP Performance. Her research interests include queuing theory, models in product-form solution and more in general stochastic modelling of networks and telecommunication systems aiming at performance and reliability evaluation and improvement.

Adityo Anggraito received the master's degree in computer science from Ca' Foscari University. He is currently working toward the graduate degree with the Ca' Foscari University of Venice. His research interests include queuing theory, and its integration with other mathematical models and simulations to solve real-world problems.

Marco Ajmone Marsan (Life Fellow, IEEE) received the honorary degree from the Budapest University of Technology and Economics. He is a research professor with the IMDEA Networks Institute in Spain. From 1974 to 2021 he was with the Politecnico di Torino, with an interruption from 1987 to 1990, when he was with the University of Milan. He is member of the Academia Europæa and of the Academy of Sciences of Torino. He is qualified as "ISI Highly Cited Researcher" in computer science.

Simonetta Balsamo is a professor of computer science with the University Caâ Foscari of Venice, Italy. Her research interests include performance and reliability modelling and analysis of computer and communication systems, parallel and distributed processing, and distributed simulation. She has published several papers in international journals, conference proceedings, book chapters, and editions of special issues of journal and international conferences. She is member of the ACM.

Andrea Marin (Senior Member, IEEE) received the PhD degree in computer science from the University of Venice in 2009. He is professor of computer science with the same university. He is the (co)author of more than 100 technical papers in refereed international journals and conference proceedings. His current research focuses on the performance and reliability evaluation of computer systems using stochastic modeling techniques.