A Black-Box Fork-Join Latency Prediction Model for Data-Intensive Applications

Minh Nguyen, Sami Alesawi, Ning Li, Hao Che, *Senior Member, IEEE*, and Hong Jiang, *Fellow, IEEE*

Abstract—The workflows of the predominant datacenter services are underlaid by various Fork-Join structures. Due to the lack of good understanding of the performance of Fork-Join structures in general, today's datacenters often operate under low resource utilization to meet stringent service level objectives (SLOs), e.g., in terms of tail and/or mean latency, for such services. Hence, to achieve high resource utilization, while meeting stringent SLOs, it is of paramount importance to be able to accurately predict the tail and/or mean latency for a broad range of Fork-Join structures of practical interests. In this article, we propose a black-box Fork-Join model that covers a wide range of Fork-Join structures for the prediction of tail and mean latency, called ForkTail and ForkMean, respectively. We derive highly computational effective, empirical expressions for tail and mean latency as functions of means and variances of task response times. Our extensive testing results based on model-based and trace-driven simulations, as well as a real-world case study in a cloud environment demonstrate that the models can consistently predict the tail and mean latency within 20 and 15 percent prediction errors at 80 and 90 percent load levels, respectively, for heavy-tailed workloads, and at any load levels for light-tailed workloads. Moreover, our sensitivity analysis demonstrates that such errors can be well compensated for with no more than 7 percent resource overprovisioning. Consequently, the proposed prediction model can be used as a powerful tool to aid the design of tail-and-mean-latency guaranteed job scheduling and resource provisioning, especially at high load, for datacenter applications.

Index Terms—Tail latency, mean response time, Fork Join queuing networks, datacenter resource provisioning

1 Introduction

Fork-Join structures underlay many datacenter services, including web searching, social networking, and big data analytics. A Fork-Join structure is a critical building block in the job processing workflow that constitutes a major part of job processing time and hardware cost, e.g., more than two-third of the total processing time and 90 percent hardware cost for a Web search engine [1]. In a Fork-Join structure (see Fig. 1), each job in an incoming flow spawns multiple tasks, which are forked to, queued and processed at different nodes, called Fork nodes in this paper, in parallel and its task results are then merged at a Join node to yield the final result. Due to barrier synchronization, the job response time is determined by the slowest task, i.e., the tail probability, which is hard to capture, from both modeling and measurement points of view, making it extremely challenging to predict the job performance, e.g., the job tail latency.

Tail latency is considered to be the most important performance measure for user-facing datacenter applications [2], such as web searching and social networking, and normally expressed as a high percentile job response time, e.g., the 99th percentile response time of 200 ms. Mean latency is also an important performance measure for big data analytics workloads which are generally scale-out by design, involving one or multiple rounds of parallel processing of a (massive) number of tasks and task result merging phases with barrier synchronization, based on, e.g., MapReduce [3] or Spark [4] frameworks. In addition, it is harder but more important[1] to predict the tail and mean latency under heavy load conditions than light ones. This is because as the load becomes heavier, so does the tail distribution, e.g., the 99th percentile of memcached request latencies on a server jumps from less than 1 ms at the load of 75 percent to 1 s at the load of 89 percent [5].

Due to the lack of good understanding of the job-vs-task performance of such workloads, i.e., how distributed task-level performance determines the job-level performance, especially in the high load[2] region, to provide high assurance of meeting tail-latency and/or mean-latency SLOs for such workloads, the current practice is to overprovision resources, which however, results in low resource utilization in datacenters [6], [7]. For example, aggregate CPU and memory utilizations in a 12,000-server Google cluster are mostly less than 50 percent, leaving 50 and 40 percent allocated CPU

- *M. Nguyen, N. Li, H. Che, and H. Jiang are with the Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington, TX 76019. E-mail: mqnguyen@mavs.uta.edu, {ning.li, hong.jiang}@uta.edu, hche@cse.uta.edu.*
- *S. Alesawi is with the Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington, TX 76019, and also with the Faculty of Computing and Information Technology in Rabigh, King Abdulaziz University, Jeddah 21589, Saudi Arabia. E-mail: salesawi@kau.edu.sa.*

Manuscript received 29 July 2019; revised 27 Feb. 2020; accepted 3 Mar. 2020. Date of publication 0 . 0000; date of current version 0 . 0000.
(Corresponding author: Minh Nguyen.)
Recommended for acceptance by jianfeng Zhan.
Digital Object Identifier no. 10.1109/TPDS.2020.2982137

1. In the low load region, tail and/or mean latency requirements can be easily satisfied as the available resources are abundant. In contrast, in the heavy load region in which the leftover resource is scarce, resource allocation with high precision must be exercised to meet user requirements.

2. The term "load" can be generally defined as the offered workload per unit time divided by processing capacity per unit time. In the context of Fork-Join structure, it is the maximum of the loads among all the Fork nodes.

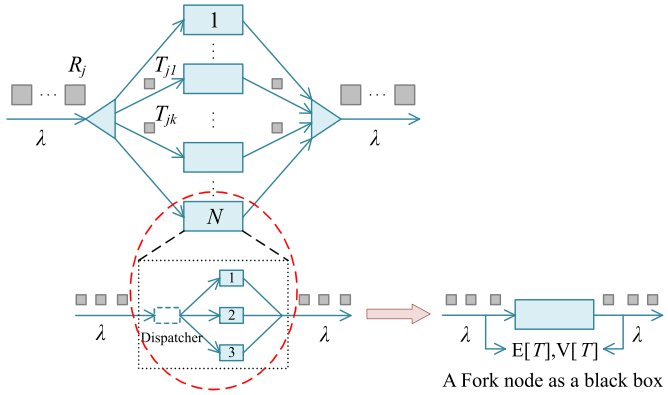


Fig. 1. Black-box Fork-Join model. Each job in the incoming flow spawns k tasks mapped to k out of N Fork nodes. Each Fork node is treated as a black box, completely determined by the mean and variance of the task response time, i.e., $\mathbb{E}[T]$ and $\mathbb{V}[T]$.

and memory resources, respectively, idle almost at all time [6]. Similarly, in a large production cluster at Twitter, aggregate CPU usage is within 20–30 percent even thought CPU reservations are up to 80 percent and aggregate memory usage is mostly within 40–50 percent while memory allocation consistently exceeds 75 percent [7]. Hence, how to improve resource utilization or the load from currently less than 50 percent to, say, 80-90 percent, while meeting stringent SLOs has been a challenging issue for datacenter service providers [7]. To this end, *a key challenge to be tackled is how to accurately capture the tail and mean latency with respect to various Fork-Join structures at high load*.

Fork-Join structures are traditionally modeled by a class of queuing networks, known as Fork-Join queuing network (FJQN) [8], as depicted in Fig. 1. FJQNs are white-box models in the sense that all the Fork nodes are explicitly modeled as queuing systems with given arrival process, queuing discipline, and service time distribution. In this paper, we argue that attempting to use FJQNs to cover a sufficiently wide range of Fork-Join structures of practical interests is not a viable solution. Instead, a black-box solution that can cover a broad range of Fork-Join structures must be sought.

On one hand, FJQNs are notoriously difficult to solve in general. Despite the great effort made for more than half a century, to date, no exact solution is available even for the simplest FJQN where all the nodes are M/M/1 queues [9], i.e., Poisson arrival process and one server with exponential service time distribution. Although empirical solutions for some FJQNs are available, e.g., [10], [11], [12], [13], [14], they can only be applied to a very limited number of Fork-Join structures, e.g., homogeneous case, the case of First-In-First-Out (FIFO) queuing discipline, and a limited number of service time distributions.

On the other hand, *the design space of Fork-Join structures of practical interests* is vast. It encompasses (a) a wide range of queuing disciplines and service time distributions (e.g., both light-tailed and heavy-tailed) [8]; (b) the case with multiple replicated servers per Fork node for failure recovery, task load balancing, and/or redundant task issues for tail cutting [15], [16] or fast recovery from straggling tasks [17]; (c) the case where the number of spawned tasks per job may vary from one job to another [18]; and (d) the case of consolidated services, where different types of services and applications may share the same datacenter cluster resources [19]. Clearly, the existing FJQNs can hardly cover such a design space in practice.

To tackle the above challenges, in this paper, we propose to study a *black-box* Fork-Join model for the prediction of job tail and mean latency, called ForkTail and ForkMean, respectively, to cover a broad range of Fork-Join structures of practical interests. By "black-box", we mean that each Fork node is treated as a black box, regardless of how many replicated servers there are and how tasks are distributed, queued, and processed inside the box. In other words, for a black-box Fork-Join model, one can only use the task statistics measurable from outside of Fork nodes, e.g., the mean and variance of the task response time (see Fig. 1). This is in stark contrast to a white-box Fork-Join model where the exact task queuing discipline and the service model for a Fork node must be known. It also allows the number of spawned tasks per job, k, to be a random integer taking values in $[1, N]$, where N is the maximum number of Fork nodes. As we shall see, our black-box model can indeed adequately covers the above design space.

However, general solutions to this model are unlikely to exist, given the limited success in solving the white-box FJQNs. Nevertheless, we found that for the black-box model, empirical solutions under heavy load conditions do exist, known as the central limit theorem for G/G/m queuing systems, where the arrival process is general with independent interarrival times, the queuing discipline is FIFO, and there are m servers with general service time distributions, under heavy load [20], [21]. Inspired by this theorem, we were able to demonstrate [22] that in a load region of 80 percent or higher, where resource provisioning with precision is most desirable and necessary, an empirical expression of the tail-latency for a special case of the black-box model, i.e., $k = N$ for all the requests, exists, which can predict the tail latencies within 15 percent error at any load levels for light-tailed service time distribution and the load level of 90 percent for heavy-tailed one in the cases (a) and (b) in the design space mentioned above. As our sensitivity analysis in Section 4 shows, such prediction errors can be well compensated for with no more than 7 percent resource overprovisioning.

The work in this paper makes the following contributions. First, it generalizes the solution in [22] to also cover cases (c) and (d) in the design space, hence, making it applicable to most Fork-Join structures of practical interests. Second, it gives the first empirical, universal solutions to tail and mean job latencies for both black-and-white-box FJQNs at high load and hence, it makes a contribution to the queuing network theory as well. In fact, for any white-box FJQN with G/G/1 Fork queuing servers, our approach leads to closed-form approximate solutions, which are on par with the most elaborate white-box solutions in terms of accuracy across the entire load range at much lower computational complexity. Third, comprehensive testing and verification of the proposed approximations for tail and mean latency are performed for all (a)–(d) Fork-Join structures, based on model-based and trace-driven simulation, as well as a real-world case study. Fourth, sensitivity analysis indicates that our proposed solutions can lead to accurate resource provisioning for data-intensive services and applications in a consolidated

datacenter environment at high load. Finally, preliminary ideas are provided as to how to use this solution to facilitate SLO-guaranteed job scheduling and resource provisioning.

The rest of the paper is organized as follows. Section 2 introduces our black-box model and ForkTail and ForkMean, the empirical approximations for the tail and mean latency, respectively. Section 3 performs extensive testing of the accuracy of these approximations. Section 4 presents the sensitivity analysis for the proposed approximations. Section 5 explores the range of applicability of the proposed solutions. Section 6 discusses how the proposed approximations may be used to facilitate effective job scheduling and resource provisioning with tail-latency-SLO guarantee. Section 7 reviews the related work. Finally, Section 8 concludes the paper and discusses future work.

2 Model and Solutions

2.1 Black-Box Model

The black-box model described in this section greatly extends the scope of the black-box model introduced in [22] to address the entire design space mentioned in Section 1.

Consider a black-box Fork-Join model with each job in the incoming flow spawning k tasks mapped to k out of N Fork nodes, as depicted in Fig. 1. The results from all k tasks are finally merged at a Join node (i.e., the triangle on the right). Jobs arrive following a random arrival process with average arrival rate λ. Each Fork node may be composed of more than one replicated servers for task-level fault tolerance, load balancing, tail-cutting, and/or straggler recovery. An example Fork node with three server replicas is depicted in Fig. 1.

The above model deals with a general case where $k \leq N$. Note that the traditional FJQNs cover only a small fraction of this design space, i.e., $k = N$, homogeneous Fork nodes with a single server per node, which is modeled as a FIFO queuing system.

General solutions to this model are unlikely to exists. Fortunately, *we are most interested in finding solutions in high load regions where precise resource provisioning is highly desirable and necessary*. There is a large body of research results in the context of queuing performance in high load regions (e.g., see [23] and the references therein). In particular, a classic result, known as the central limit theorem for heavy traffic queuing systems [20], [21], states that for a G/G/m queue under heavy load, the waiting time distribution can be approximated by an exponential distribution. Clearly, this theorem applies to the response time distribution as well, since the response time distribution converges to the waiting time distribution as the traffic load increases. Inspired by this result, we postulate that for tasks mapped to a black-box Fork node and in a high load region, the task response time distribution $F_T(x)$ for any arrival process and service time distribution can be approximated as a generalized exponential distribution function [24], as follows,

$$F_T(x) = (1 - e^{-x/\beta})^{\alpha}, \quad x > 0, \alpha > 0, \beta > 0, \tag{1}$$

where α and β are shape and scale parameters, respectively.

The mean and variance of the task response time are given by [24]

$$\mathbb{E}[T] = \beta[\psi(\alpha + 1) - \psi(1)], \tag{2}$$

$$\mathbb{V}[T] = \beta^2[\psi'(1) - \psi'(\alpha + 1)], \tag{3}$$

where $\psi(.)$ and its derivative are the digamma and polygamma functions.

From Eqs. (2) and (3), it is clear that the distribution in Eq. (1) is completely determined by the mean and variance of the task response time. In other words, the task response time distribution can be measured by treating each Fork node as a black box as shown in Fig. 1. The rationale behind the use of this distribution, instead of the exponential distribution, is that it can capture both heavy-tailed and light-tailed task behaviors depending on the parameter settings and meanwhile, it degenerates to the exponential distribution at $\alpha = 1$ and $\mathbb{E}[T] = \beta$. In [22], we showed that this distribution significantly outperforms the exponential distribution in terms of tail latency predictive accuracy.

Now, with all the Fork nodes in Fig. 1 being viewed as black boxes, the response time distribution for any job with k tasks can be approximated using the order statistics [9] as follows,

$$F_X^{(k)}(x) = \prod_{i=1}^{k} F_{T_i}(x) = \prod_{i=1}^{k} (1 - e^{-x/\beta_i})^{\alpha_i}. \tag{4}$$

Note that the above expression is exact if the response times for tasks mapped to different Fork nodes are independent random variables. This, however, does not hold true for any Fork-Join structures, simply because the sample paths of the task arrivals at different Fork nodes are exactly the same, not independent of one another. This is the root cause that renders the Fork-Join models extremely difficult to solve in general. In what follows, we introduce ForkTail and ForkMean, separately, based on this approximation.

2.2 ForkTail

ForkTail was originally presented in [25]. Our postulation is that as load reaches 80 percent or higher where precise resource provisioning is desirable and necessary, the tail-latency prediction errors introduced by the above assumption will become small enough for resource provisioning purpose. Our extensive testing results in this paper provide strong support of the postulation, making our modeling approach the only practically viable one for tail latency prediction.

Tail latency x_p, defined as the pth percentile job response time, can be written as,

$$x_p = {F_X^{(k)}}^{-1}(p/100). \tag{5}$$

Eq. (5) simply states that in a high load region, the tail latency can be approximated as a function of the means and variances of task response times for all k tasks at their corresponding Fork nodes, irrespective of what workloads cause the heavy load. The implication of this is significant. It means that this expression is applicable to a consolidated datacenter cluster where more than one service/application share the same cluster resources. Moreover, this expression allows tail latency to be predicted using a limited number of job samples thanks to its dependence on the first two moments of task response times only, i.e., the means and variances.

The results so far is general, applying to the heterogeneous case, where task response time distributions may be

different from one task to another, due to, e.g., the use of heterogeneous Fork nodes and/or uneven background workloads. As a result, the tail latency predicted by Eq. (5) may be different from one job to another or even for two identical jobs, as long as their respective Fork nodes do not completely coincide with one another, or they are issued at different times. In other words, Eq. (5) is a fine-grained tail latency expression. For certain applications, such as offline resource provisioning (see Section 6 for explanations) and coarse-grained, per-service-based tail-latency prediction, one may be more interested in the homogeneous case only. In this case, the response time distribution can be further simplified as,

$$F_X^{(k)}(x) = (1 - e^{-x/\beta})^{k\alpha}. \tag{6}$$

This is because the means and variances given in Eqs. (2) and (3) are the same for the homogeneous case. A coarser-grained cumulative distribution function (CDF) of the job response time can then be written as,

$$F_X(x) = \sum_{k_i} F_{X|K}(x|k_i)P(K = k_i), \tag{7}$$

where $F_{X|K}(x|k_i)$ is the conditional CDF of the job response time for jobs with k_i tasks, given by Eq. (6), i.e., $F_{X|K}(x|k_i) = F_X^{(k_i)}(x)$, and $P(K = k_i) = P_i$ is the probability that a job spawns k_i tasks.

Further assume that there are m job groups with distinct numbers of tasks k_i's, $i = 1, \ldots, m$, and corresponding probabilities P_i's. We then have,

$$F_X(x) = \sum_{i=1}^{m} P_i \cdot F_X^{(k_i)}(x). \tag{8}$$

Correspondingly, the tail latency for the m groups of jobs as a whole can then be readily obtained, similar to Eq. (5), as follows,

$$x_p = F_X^{-1}(p/100). \tag{9}$$

For example, the tail latency for a given service can be predicted by collecting statistics for k_i's and P_i's, as well as mean and variance of task response time and applying them to the tail latency expression in Eq. (9).

2.2.1 Application to White-Box FJQNs

Clearly, the above black-box approach leads to closed-form solutions for any white-box models whose analytical expressions for the means and variances of task response times are available, whether it is homogeneous or not. In fact, our solution works for the case where different Fork nodes may have different service time distributions and queuing disciplines. For instance, our approach can be applied to a large class of FJQNs, where each Fork node is an M/G/1 queue or a more general G/G/1 queue, whose mean and variance of the task response time can be computed from Takács recurrence theorem [26] or the queuing network analyzer [27], respectively.

2.3 ForkMean

While the approximations in Eqs. (5) and (9) work well for the job tail latency even for the $k < N$ cases, it fails to accurately predict the job mean response time,[3] yielding more than three times larger errors for the same cases studied, especially for the case of light-tailed service time distributions. We find that the reason for this to happen is due to the fact that to accurately predict the job mean response time, the entire job response time distribution including the tail portion must be accurately captured, as the barrier synchronization tends to push the job mean response time towards the tail part of the task response time distribution, as the workload scales out.

On the basis of the above modeling, this section aims at finding solutions to reduce the prediction errors for the job mean latency. To this end, we make the following two key observations.

Observation 1. For a wide range of Fork-Join models, the difference between the exact tail-mean ratio and the model-based tail-mean ratio, derived from the CDF in Eq. (4), hereafter called the *gap* and denoted as Δ, converges to a constant as the number of Fork nodes becomes large enough. Mathematically, we have,

$$\frac{x_p}{x_m} - \frac{x_p^{\text{ge}}}{x_m^{\text{ge}}} = \Delta, \tag{10}$$

where x_p and x_m are the exact pth percentile and mean of job latency, respectively, which can be estimated by experiments, while x_p^{ge} and x_m^{ge} are derived from the prediction model, i.e., Eq. (4). Hence, the mean latency can be approximated as follows,

$$x_m = \frac{x_p}{R^{\text{ge}} + \Delta} \approx \frac{x_p^{\text{ge}}}{R^{\text{ge}} + \Delta}\ , \tag{11}$$

where $x_p \approx x_p^{\text{ge}}$ at high loads, since ForkTail give accurate predictions for the pth percentile at high loads, as indicated in the testing results, and $R^{\text{ge}} = x_p^{\text{ge}}/x_m^{\text{ge}}$.

Fig. 2 illustrates the gaps for systems with different task service time distributions, including light-tailed and heavy-tailed ones, where each Fork node is a single server, i.e., without replication. As one can see, the gap converges to a constant as N becomes sufficiently large, say, $N \geq 100$, for all the cases. Similar trends are also observed for the systems with 3-replica Fork nodes with Round-Robin and redundant-task-issue dispatching policies as well as the systems with variable numbers of forked tasks (not shown here).

Observation 2. There is a strong correlation between the tail heaviness of service time distribution and the gap Δ, i.e., the heavier the tail, the smaller the gap. It is evident from Fig. 2 that the light-tailed distributions, including Exponential and Weibull, have larger gaps than the heavy-tailed ones, including the truncated Pareto and empirical (defined in Section 3.1). With this observation, we make the following postulation: The gap is much more of a function of the tail heaviness of a service time distribution than the service time distribution itself.

From the above observations, we propose two empirical solutions, one is white-box and the other black-box, for the approximation of the gap, Δ, and hence, the job mean response time.

3. We use the terms 'latency' and 'response time' interchangely in this paper.

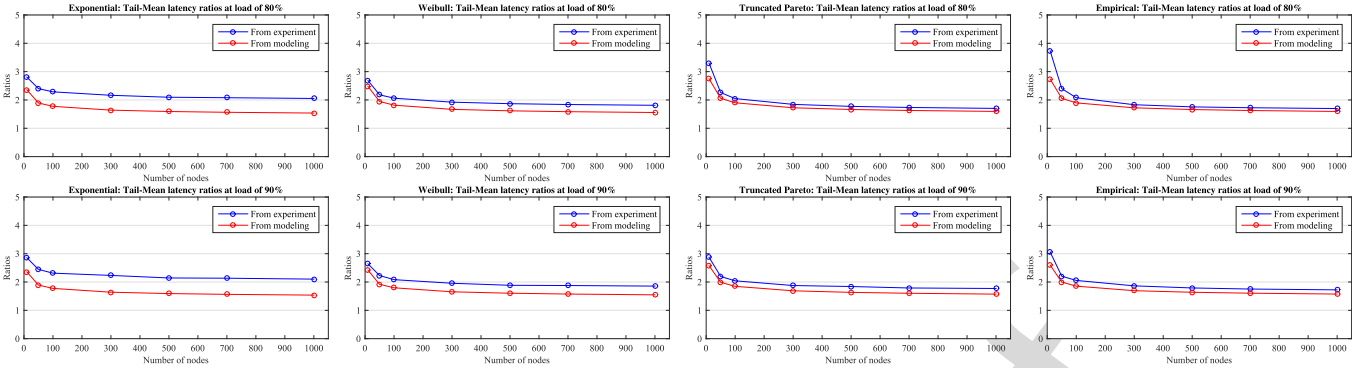


Fig. 2. The gaps for Fork-Join systems with different service time distributions at load levels of 80 percent (upper row) and 90 percent (lower row).

2.3.1 White-Box Approach

This approach is based on the above postulation. Here we consider a homogeneous white-box Fork-Join queuing model where each Fork node can be modeled as a G/G/1 queue. With known interarrival and service time distributions, one can find the job response time distribution and the corresponding tail and mean latencies, and so their ratio R^{ge}, from ForkTail. So to find the job mean latency, x_m, all that is left to be done is to find Δ.

To this end, we first define tail heaviness, $w(F_T)$. We use Right Quantile Weight [28] which measures the tail heaviness on the right side of a distribution, the region of interest in all of our experiments. This tail weight measure is defined as,

$$w(F_T) = \frac{F_T^{-1}\left(\frac{1+q}{2}\right) + F_T^{-1}\left(1-\frac{q}{2}\right) - 2F_T^{-1}(0.75)}{F_T^{-1}\left(\frac{1+q}{2}\right) - F_T^{-1}\left(1-\frac{q}{2}\right)}, \tag{12}$$

where $0.5 < q < 1$ and $F_T^{-1}(q)$ is quantile q of task service time distribution F_T. To capture the tail effect but still retain a reasonable robustness, we set $q = 0.99$.

Based on our postulation, $\Delta = \Delta(\rho, w)$, independent of $F_T(x)$. Here ρ is the load. In other words, as long as $w(F_T^{(1)}) = w(F_T^{(2)})$, the two homogeneous Fork-Join models with different service time distributions, $F_T^{(1)}$ and $F_T^{(2)}$, respectively, will have the same gap. In other words, if one can find the function, $\Delta(\rho, w)$, using one distribution function with different tail weights, this $\Delta(\rho, w)$ can then be used by any Fork-Join models with other distribution functions to find the gap. In this paper, we use the generalized exponential distribution in Eq. (1) at different coefficients of variance to generate different tail weights from Eq. (12) and the corresponding gaps and then use nonlinear regression to find $\Delta(\rho, w)$. Table 1 shows the gaps for different tail weights, averaged over $N = 100$ to 1,000 at three different load levels.

From experimental data with different distribution parammeters, we found that the power function, i.e., $\Delta = aw^b + c$, yields a very good fit to these gap-tail weight points. Fig. 3 illustrates the fitted curve at load level of 80 percent from Table 1 with respect to the fitted points from the generalized exponential distribution (the black points). It also shows the actual points from other distributions, which are used for testing in the experiments (the green points), relative to the fitted curve. As one can see, the green points stay reasonably close to the curve itself, meaning that our postulation indeed holds true. Table 2 presents the fitted functions for the cases in Table 1.

In summary, this white-box approach results in a closed-form solution for the approximation of job mean latency, which is composed of the following computation steps,

- With given $\mathbb{E}[T]$ and $\mathbb{V}[T]$, compute the tail and mean latencies, i.e., x_p^{ge} and x_m^{ge} from the predicted CDF in Eq. (4) and their corresponding ratio, i.e., R^{ge};
- With a given service time distribution F_T, calculate the tail weight w from Eq. (12), which is then mapped to a Δ at a given load, e.g., using one of the functions in Table 2;
- Approximate the mean latency using Eq. (11).

2.3.2 Black-Box Approach

The white-box approach above leads to closed-form solutions for homogeneous white-box Fork-Join models with known

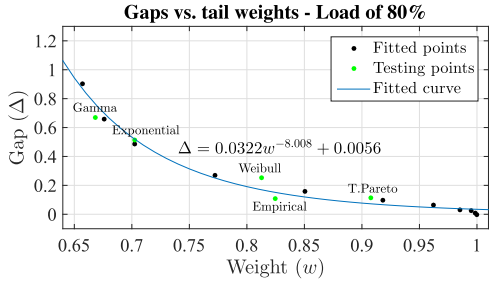


Fig. 3. An example of the gap-vs-tail-weight fitted curve.

TABLE 1
The Gaps for Different Tail Heavinesses and Load Levels

Load	Tail weight						
	0.703	0.772	0.851	0.918	0.962	0.986	0.999
75%	0.486	0.271	0.160	0.097	0.063	0.029	0.009
80%	0.511	0.283	0.169	0.106	0.069	0.044	0.013
90%	0.573	0.319	0.190	0.129	0.070	0.055	0.023

TABLE 2
Examples of Fitted $\Delta(\rho, w)$ Curves

Load	Function
75%	$\Delta = 0.0371w^{-7.517} - 0.0052$
80%	$\Delta = 0.0322w^{-8.008} + 0.0056$
90%	$\Delta = 0.0274w^{-8.654} + 0.0284$

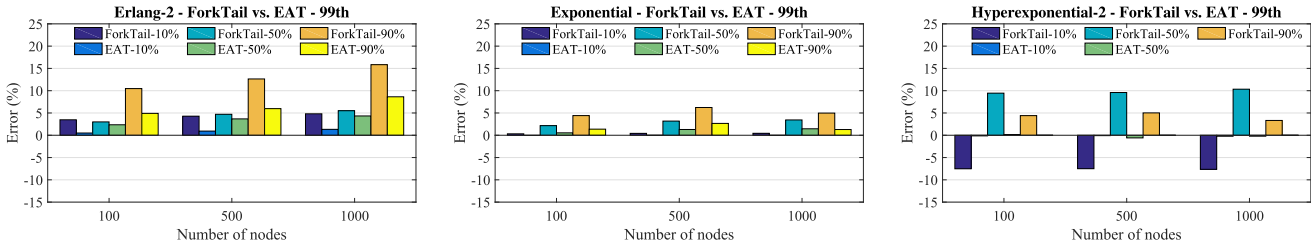


Fig. 4. Prediction errors for the 99th percentile response times for ForkTail and EAT.

service time distributions for Fork nodes. However, in practice, determining those distributions is nontrivial, e.g., for systems with multi-replica Fork nodes. Hence, it is necessary to seek a black-box solution applicable to a wide range of Fork-Join structures of practical interests.

Based on the Observation 1, i.e., Δ converges to a constant as the number of Fork nodes becomes large enough, i.e., around 100, based on all the testing cases. This suggests that, if for a target application, Δ can be measured on a small testbed or by simulation, with 100 virtual machines/nodes, or equivalently, a few commodity servers, e.g., 5, then the mean latency can be predicted when the application is deployed on a much larger number of nodes. This approach requires only the means and variances of task response times as inputs, and hence is a hybrid, black-box solution.

The steps taken to find the job mean latency are similar to those for the white-box approach above except for step 2 where Δ is predicted by running experiments for the target application on a system with a given number of Fork nodes, e.g., 100, and measure the ratio gap between the results from the experiments and the prediction model.

Compared to the white-box solution, the black-box one is simpler and can be applied to a much wider range of Fork-Join structures. However, as a hybrid approach, it requires to run experiments, either via simulation or on a real testbed, with an adequate number of Fork nodes, e.g., 100. Consequently, it should be applied to large-scale systems where a job is forked to at least hundreds of nodes, much larger than the one used for testing. Note that the hybrid approach, which combines analysis and simulation, is not unusual in analyzing performance of the Fork-Join model. Indeed, it has been used in several previous works in the literature [10], [13], [29].

3 Validation

3.1 Tail Latency Prediction Validation

In this section, ForkTail is extensively validated against the results from model-based simulation, trace-driven simulation, and a case study in Amazon EC2 cloud. The validation is performed for the systems with $k = N$, $k \leq N$, and consolidated services, separately. The accuracy of the prediction is measured by the relative error between the value predicted from ForkTail, t_p, and the one measured from simulation or real-system testing, t_m, i.e.,

$$error = \frac{100(t_p - t_m)}{t_m}.$$

3.1.1 *Case 1:* $k = N$

A notable example for this case is Web search engine [30] where a search request looks up keywords in a large inverted index distributed on all the servers in the cluster. We validate ForkTail with three testing approaches, i.e., white-box and black-box model-based testing as well as a real-world case study in Amazon EC2 cloud.

White-Box Model-Based Validation. Here we study the accuracy of ForkTail when applied to homogeneous, single-queuing-server-Fork-node Fork-Join systems with the assumption that the service time distribution is known in advance, the approach taken in all the existing works on performance analysis of FJQNs [9]. The tail latency prediction involves the following steps:

- Find the mean and variance of task response times with the given task service time distribution;
- Substitute the above mean and variance into Eqs. (2) and (3), respectively, and solve that system of equations to find the scale and shape parameters of the generalized exponential distribution in Eq. (1), which is then used to approximate the task response time distribution;
- Calculate the pth percentile of request response times from Eq. (9).

First, we compare ForkTail against the state-of-the-art tail latency approximation for *homogeneous* FJQNs [14], known as *EAT*, which is derived from analytical results for single-node and two-node systems. Fig. 4 shows the comparative results for three service time distributions studied in [14], i.e., Erlang-2, Exponential, and Hyperexponential-2, at the loads of 10, 50, and 90 percent[4] and numbers of nodes of 100, 500, and 1,000.

EAT provides more accurate (from a few to several percentage points) approximations for the 99th percentiles of response times across all the cases studied. Much to our surprise, our approach yields most of the errors within 10 percent, across the entire load range. Although outperforming our approach, EAT has its limitations. First, it can be applied only to homogeneous FJQNs where each node can be generally modeled as a MAP/PH/1 queuing system, i.e., Markovian arrival processes and phase-type service time distribution with one service center. Second, the method requires the service time distribution to be known in advance and converted into a phase-type distribution, which is nontrivial, especially for heavy-tailed distributions [31]. Third, the method may incur high computational complexity, depending on the selection of a constant C, whose value determines the computational runtime and prediction accuracy. It takes

4. For EAT, the case for Hyperexponential-2 at the load of 90 percent is not available, due to a numerical error running the code provided in [14].

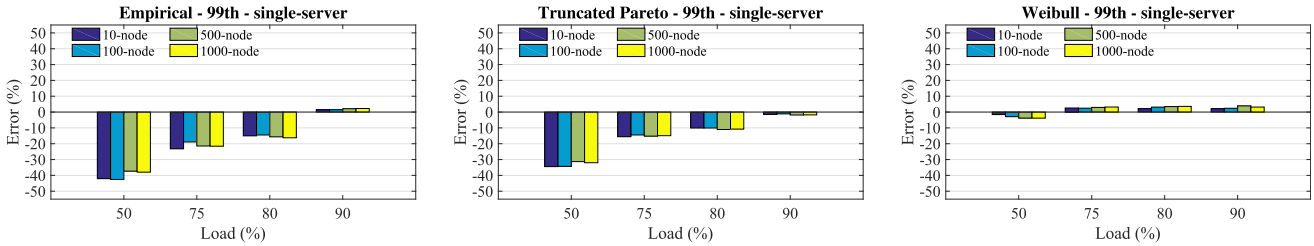


Fig. 5. Prediction errors of the 99th percentile response times for white-box systems with single-server Fork nodes.

at least 2 seconds on our testing PC (Core i7-4940MX Quad-core, 32GB RAM) to get the resulting percentiles even at the lesser degree of accuracy with $C = 100$ (more than 300 seconds at $C = 500$). In contrast, our method takes less than 5 milliseconds to compute the required percentiles. As a result, similar to other existing white-box solutions, EAT has limited applicability for datacenter job scheduling and resource provisioning in practice.

To cover a sufficiently large workload space, we further consider service time distributions with heavy tails, which are common in practice [32] and cannot be easily dealt with by EAT, including the following,

- Empirical distribution measured from a Google search test leaf node provided in [32], which has a mean service time of 4.22 ms, a coefficient of variance (CV) of 1.12, and the largest tail value of 276.6 ms;
- Truncated Pareto distribution [31] with the same mean service time and a CV of 1.2, whose CDF is given by,

$$F_S(x) = \frac{1-(L/x)^{\alpha}}{1-(L/H)^{\alpha}} \qquad 0 \le L \le x \le H, \tag{13}$$

  where α is the shape parameter; L is the lower bound; and H is the upper bound, which is set at the maximum value of the empirical distribution above, i.e., $H = 276.6$ ms, resulting in $\alpha = 2.0119$ and $L = 2.14$ ms.
- Weibull distribution [8], also with the same mean service time and a CV of 1.5, whose CDF is defined as,

$$F_S(x) = 1 - \exp[-(x/\beta)^{\alpha}] \qquad x \ge 0, \tag{14}$$

  where $\alpha = 0.6848$ and $\beta = 3.2630$ are shape and scale parameters, respectively.

Fig. 5 presents the prediction errors for the 99th percentile response times for the above cases. The Weibull distribution, which is less heavy-tailed, consistently yields smaller errors, well within 5 percent, for the entire load range studied, similar to the light-tailed distribution cases studied earlier. The empirical and truncated Pareto distributions, which are more heavy-tailed, provide good approximations for the 99th percentiles at the load of 80 percent or higher, which is well within 17 and 5 percent at the load of 80 and 90 percent, respectively, agreeing with our postulation.

We also consider the cases with general arrival process and general service time distribution, i.e., G/G/1 Fork nodes. Fig. 6 shows the prediction errors for example cases with Erlang-2 (CV = 0.5) and Hyperexponential-2 (CV = 1.2) arrival processes and Truncated Pareto service time distribution (CV = 3.0). Again, ForkTail yields quite accurate approximations for tail latency at high load regions, i.e., above 75 percent. The prediction results also show the same trend for Weibull and Exponential service time distributions, which are not shown here.

Black-Box Model-Based Validation. We now validate ForkTail without making assumption on the service time distribution at each Fork node. We treat each Fork node as a black-box and empirically measure the mean and variance of task response times at each given arrival rate λ or load. These measures are then substituted into Eqs. (2) and (3), respectively, to find the shape and scale parameters, which are in turn used to predict the tail latency based on Eq. (9).

For all the three heavy-tailed FJQNs studied above, we consider two types of Fork nodes, i.e., one with single server and the other with three replicated servers. For the one with three servers, we explore two task dispatching policies. The first policy is the Round-Robin (RR) policy, in which the dispatcher will send tasks to different server replicas in an RR fashion. The second policy is still RR, but it also allows redundant task issues, a well-known tail-cutting technique [15], [16]. This policy allows one or more replications of a task to be sent to different server replicas in the Fork node. The replications may be sent in predetermined intervals to avoid overloading the server replicas. In our experiments, at most one task replication can be issued, provided that the original one does not finish within 10 ms, which is around the 95th percentile of the empirical distribution above.

Figs. 7, 8, and 9 present the prediction errors at different load levels and N's for the 99th percentile response times for all three FJQNs with single server and three servers per Fork node, respectively. First, we note that the prediction errors for the cases in Fig. 7 are very close to those in Fig. 5. This is expected as the white-box and black-box results, ideally, should be identical. The differences are introduced due to simulation and measurement errors. Second, the prediction performances of the cases with three replicas and the RR policy in Fig. 8 are also very close to those of the cases in Fig. 7, with errors being well within 20 and 10 percent at the

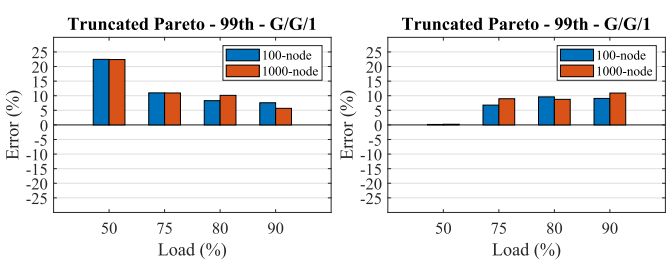


Fig. 6. Prediction errors of the 99th percentile response times for white-box systems with Erlang-2 (left) and Hyperexponential-2 (right) arrival distributions and Truncated Pareto service time distribution.

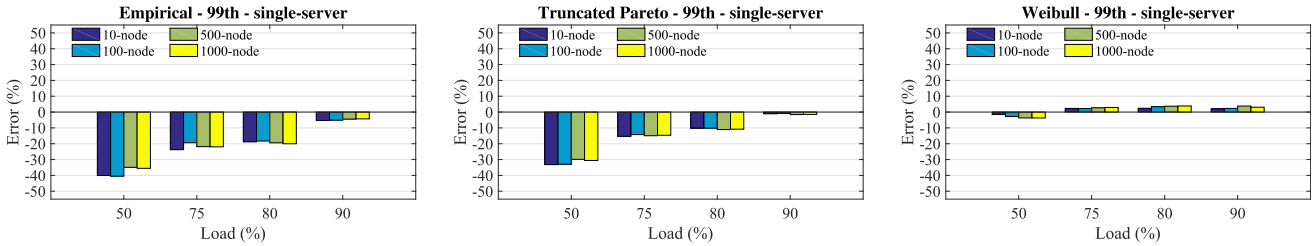


Fig. 7. Prediction errors of the 99th percentile response times for black-box systems with single-server Fork nodes.

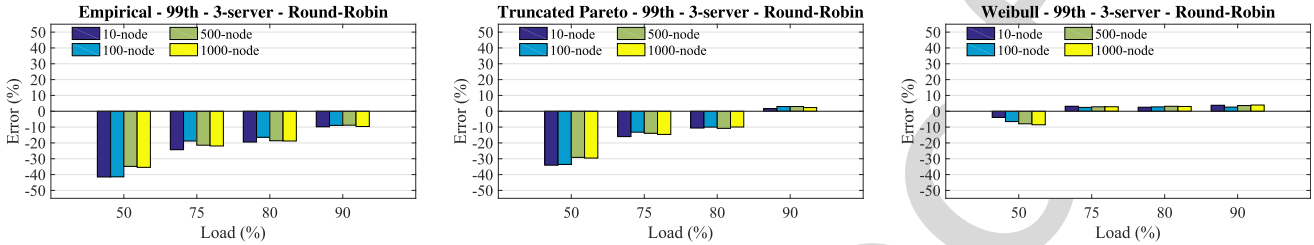


Fig. 8. Prediction errors of the 99th percentile response times for black-box systems with 3-server Fork nodes and Round-Robin policy.

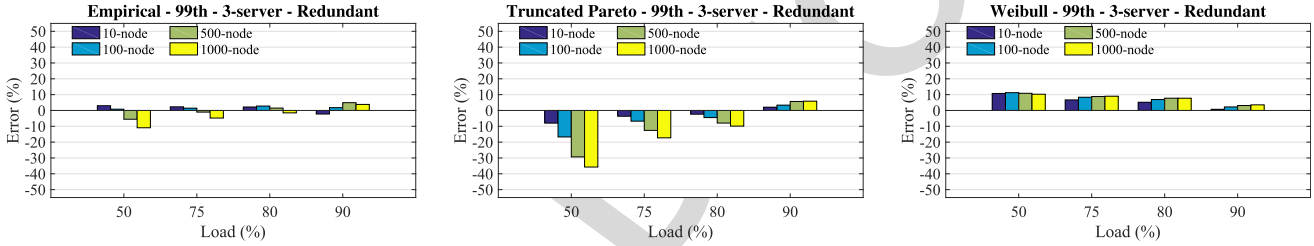


Fig. 9. Prediction errors of the 99th percentile response times for black-box systems with 3-server Fork nodes and redundant-task-issue policy.

loads of 80 and 90 percent, respectively, for all the case studies, further affirming our postulation. The two scenarios have similar performance because they are compared at the same load levels, where the RR policy in the second scenario simply balances the load among three replicas, making each virtually identical to the single-server scenario. In contrast to these two scenarios, Fig. 9 shows that with the application of the tail-cutting technique, the prediction errors are substantially reduced, with less than 10 percent at the load of 80 percent or higher. This is consistent with the earlier observation, i.e., the lighter the tail, the smaller the prediction errors. This suggests that the tail-cutting techniques, often utilized in datacenters to curb the tail effects, can help expand the load ranges in which ForkTail can be applied.

A Case Study in Cloud. We also assess the accuracy of ForkTail for a real case study in Amazon EC2 cloud. We implement a simple Unix grep-like program on the Apache Spark framework (version 2.1.0) [4]. It looks up a keyword in a set of documents and returns the total number of lines containing that keyword, as depicted in Fig. 10. The cluster for the testing includes one master node using an EC2 c4.4xlarge instance and 32 or 64 worker nodes using EC2 c4.large instances. We use a subset of the English version of Wikipedia as the document for lookup. Each worker node holds a shard of the document whose size is 128 MB, corresponding to the default block size on Hadoop Distributed File System (HDFS) [33]. A client, which runs a driver program, sends a flow of keywords, each randomly sampled from a pool of 50K keywords, to the testing cluster for lookup. Each worker searches through its corresponding data block to find the requested keyword and counts the number of lines containing the keyword. The line count is then sent back to the client program to sum up. Clearly, this testing setup matches the black-box model.

We measure the request response time, i.e., the time it takes to finish processing each keyword at the client. We also collect the task response times, composed of the task waiting time and task service time. The task waiting time is the one between the time the request the task belongs to is sent to the cluster and the time the task is sent to a given worker for processing. This is because in the Spark framework, all the tasks spawned by a request are kept in their respective

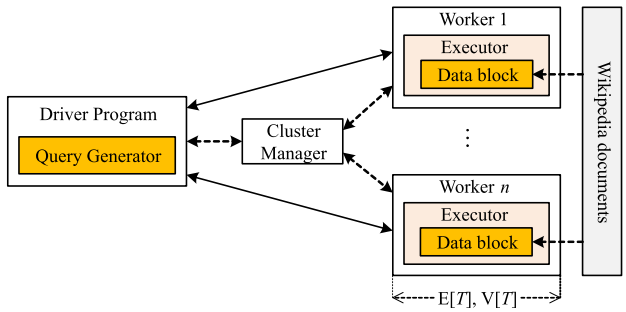


Fig. 10. Experiment setup in Amazon EC2 cloud. Each worker should be viewed as a blackbox as in Fig. 1.

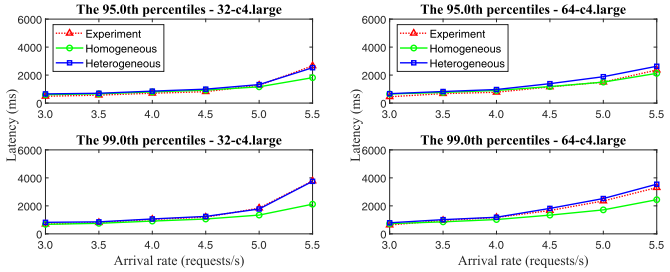


Fig. 11. Predicted tail latencies for keyword occurrence counts in Amazon cloud with 32 (left) and 64 (right) nodes.

TABLE 3
Estimated Loads (%) for the Testbed Based on Request Arrival Rates

#workers	Request arrival rates (requests/s)					
	3.0	3.5	4.0	4.5	5.0	5.5
32	48.33	56.39	64.44	72.50	80.56	88.61
64	50.04	58.38	66.72	75.06	83.40	91.74

virtual queues corresponding to their target workers centrally. A task at the head of a virtual queue cannot be sent to its target worker until the worker becomes idle. Hence, to match our black-box model, the task response time must include the task waiting time, i.e., the task queuing time plus the task dispatching time, and the task service time, which is the actual processing time at the worker the task is mapped to. From the collected samples, we compute the means and variances of task response times, which are in turn used to derive the task response time distribution as in Eq. (1).

Ideally, the task response time distributions for all the tasks are the same, given that the workers are identical. In other words, one would expect that this case study is homogeneous. However, our measurement indicates otherwise. A careful analysis reveals that this is mainly due to the task scheduling mechanism in the Spark framework. Each data block has three replicas distributed across different workers. By default, the placement preference is to send a task to an available worker where the data block resides. Unfortunately, as the request arrival rate or load increases, more tasks are mapped to workers that do not hold the required data blocks for the tasks, causing long task response time due to the need to fetch the required data blocks from the distributed file system. This results in higher variability in the task response time distributions among different workers. Therefore, the heterogeneous model given in Eq. (4) is found to be more appropriate in high load regions.

The above observation is confirmed by the experimental results, presented in Fig. 11. As one can see, the heterogeneous model (the blue lines) gives quite accurate prediction for both 95th and 99th percentiles at both $N = 32$ and 64 cases, while the prediction from the homogeneous model (the green lines) gets worse as the load becomes higher. Based on the heterogeneous prediction, the prediction errors at both $N = 32$ and 64 and the 99th percentile are well within 10 percent in a high load region, i.e., 60 percent or higher. Note that the load here is measured in terms of request arrival rate. Since the system is heterogeneous, we estimated the equivalent loads corresponding to different arrival rates based on the maximum value of means of task service times across all the workers, as given in Table 3.

Finally, we note that to achieve a reasonably good confidence of measurement accuracy for the 99th percentile tail latency, we collected 80K samples in our experiments at the maximum possible sampling rate equal to the average request arrival rate of 5.8 per second, which translates into a measurement time of 13,793 seconds or about 4 hours. It takes even more time to run the experiments at lower arrival rates. The average runtime across all the request arrival rates in the experiments is about 6 hours. Due to the costly cloud services, we have to limit our experiments to 64 worker nodes.

This example clearly demonstrates that it can be expensive and time consuming, if practical at all, to estimate tail latency based on direct measurement. In contrast, ForkTail is able to do so with far fewer number of samples at much lower cost. For example, with 800 samples collectable in less than three minutes, we can estimate the response-time means and variances for all the tasks and hence the tail latency with reasonably good accuracy. This means that our prediction model can reduce the needed samples or prediction time by two orders of magnitude than the direct measurement.

3.1.2 *Case 2: Variable Number of Tasks* $k \leq N$

Notable examples for this case are key-value store systems in which a key lookup may touch only a partial number of servers and web rendering which requires to receive web objects or data from a group of servers in a cluster.

In this case study, we assess the accuracy of our prediction model (i.e., Eqs. (8) and (9)) for applications whose jobs may spawn different numbers of tasks with distribution $P(K = k_i)$. Specifically, we study two scenarios where $P(K = k_i)$ is non-zero for a specific value of K and uniformly distributed, respectively. We further consider three different service time distributions: two heavy-tailed ones, the empirical and truncated Pareto as in Section. 3.1.1, and a light-tailed exponential distribution, with the same mean service time, i.e., 4.22 ms.

Scenario 1: Fixed Number of Tasks per Job. In this scenario, we consider the cases when the number of forked tasks per job is a fixed number k ($k \leq N$), i.e., every incoming job is

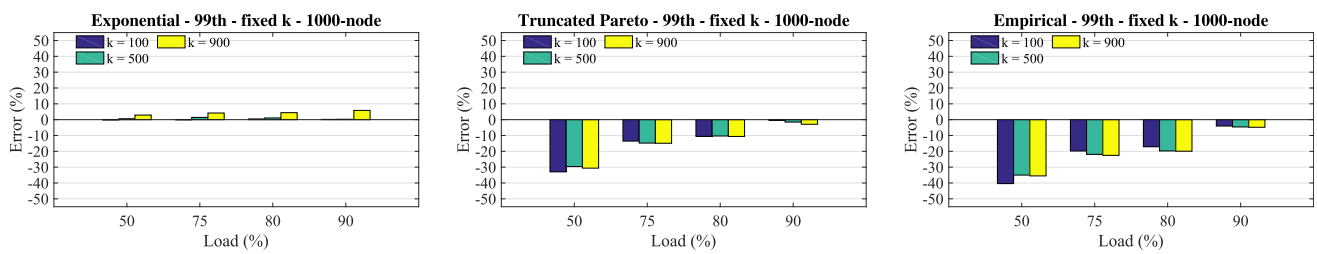


Fig. 12. Prediction errors of the 99th percentile response times for an 1000-node cluster when the number of tasks per job is fixed ($k = 100, 500, 900$).

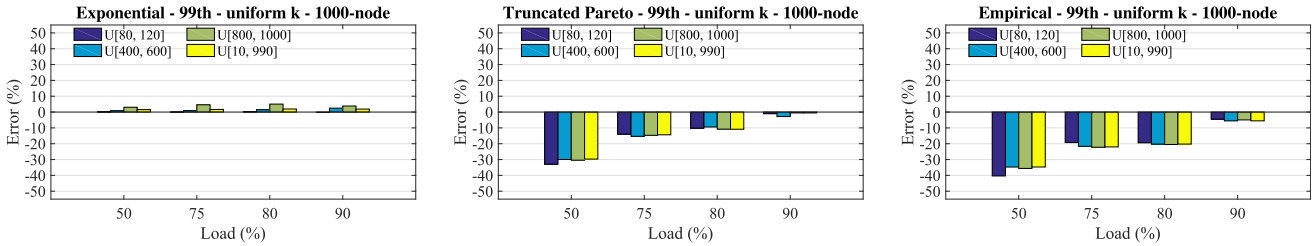


Fig. 13. Prediction errors of the 99th percentile response times for an 1000-node cluster when the number of tasks per job is uniformly distributed.

split into exactly k tasks which are dispatched to k randomly selected Fork nodes in an N-node cluster.

Fig. 12 shows prediction errors for the 99th percentile response times for an 1,000-node cluster with $k = 100$, 500, and 900 tasks. ForkTail provides good prediction in high load regions, with all the errors within 10 percent at the load of 90 and 20 percent at the load of 80 percent for all the cases studied. The case with the light-tailed exponential distribution gives quite accurate prediction for the entire range under study, i.e., all within 6 percent.

Scenario 2: Uniform Distribution. Here we deal with cases when an incoming job is forked to k random nodes in the cluster where k is randomly sampled from an integer range $[a, b]$, i.e., $k_i \in \{a, a+1, \ldots, b-1, b\}$ with probability $P_i = P = 1/m \forall i$, where $m = b - a + 1$. Therefore, the mean number of tasks is $(a + b)/2$.

Fig. 13 presents prediction errors for an 1,000-node cluster with k in four different ranges, i.e., [80, 120], [400, 600], [800, 1000], and [10, 990]. The results again show that ForkTail yields good approximations for the 99th percentile job response times when the system is under heavy load, i.e., 80 percent or higher. Furthermore, again for all the cases with the exponential distribution, ForkTail gives accurate predictions across the entire load range studied.

The above prediction model applies to the case where a single tail-latency SLO is imposed on a service or application as a whole, a practice widely adopted in industry. However, this practice can be too coarse grained. To see why this is true, Table 4 provides the predicted tail latencies for some given jobs with distinct k values in a cluster of size 1,000 and at the load of 90 percent. As one can see, the 99th percentile tail latencies for jobs at different k's can be drastically different, e.g., the 10-task and 900-task cases. This suggests that even for a single application, finer grained tail latency SLOs may need to be enforced to be effective, e.g., enforcing tail-latency SLOs for job groups with each having k's in a small range. Table 5 shows that ForkTail can indeed provide accurate, finest-grained prediction at given k's, i.e., all well within 10 percent at load of 90 percent.

TABLE 4
The Predicted 99th Percentile of Latencies (ms)

Distribution	Number of forked tasks				
	10	400	500	600	900
Exponential	291.32	446.97	456.38	464.08	481.19
Truncated Pareto	448.83	705.45	720.97	733.66	761.87
Empirical	391.27	616.22	629.83	640.95	665.68

TABLE 5
Errors in the 99th Percentile Prediction When Tracking Jobs With a Given Number of Tasks at Load of 90 percent

Distribution	Number of nodes				
	10	400	500	600	900
Exponential	−0.861	0.052	0.433	0.647	2.791
Truncated Pareto	−0.571	−0.403	1.763	−0.489	−1.433
Empirical	−2.814	−6.929	−6.239	−5.322	−6.541

3.1.3 Case 3: Consolidated Services

In this case study, we evaluate the accuracy of ForkTail when applied to the consolidated datacenter where multiple applications, including latency-sensitive user-facing and background batch ones, share cluster resources as illlustrated in Fig. 14. We conduct a trace-driven simulation based on a trace file derived from the Facebook 2010 trace, a widely adopted approach in the literature to explore datacenter workloads [19], [34], [35]. We test the accuracy of ForkTail in capturing the tail latency for a given *target application*.

Workload. The trace file is generated based on the description of the Facebook trace in some previously published works [19], [34], [35]. Specifically, we first generate the number of tasks for job arrivals based on the distribution of the job size in terms of the number of tasks per job, as suggested in [35]. It includes nine bins of given ranges of the number of tasks and corresponding probabilities, assuming that the number of tasks is uniformly distributed in the range of each bin. We then generate the mean task service time based on the Forked task processing time information in [34]. Individual task times are drawn from a Normal distribution with the generated mean and a standard deviation that doubles the mean as in [19]. The resulting trace file contains a total of two million requests, each including the following information: request arrival time, number of forked tasks, mean task service time, and the service times of individual forked tasks.

In the experiments, the jobs in the trace file serve as the background workloads, which are highly diverse, involving

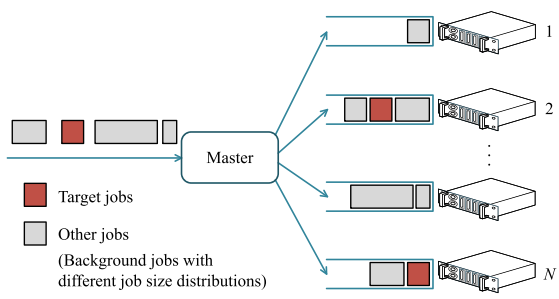


Fig. 14. Consolidated applications running on a cluster.

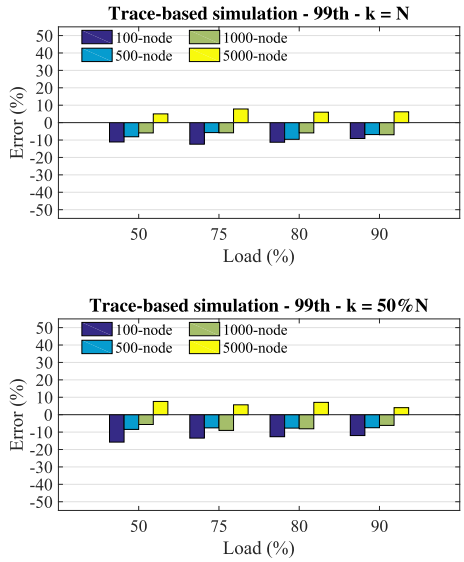


Fig. 15. Prediction errors of the 99th percentile target response times in a consolidated workload environment when the tasks of each target job reach all the nodes (top) and randomly reach 50 percent number of nodes (bottom) in the cluster.

a wide range of applications with mean service times ranging from a few milliseconds to thousands of seconds. The target jobs are generated at runtime using the same approach the trace file is generated. The only difference is that the target jobs are statistically similar with the same mean service time, to mimic a given application or simply a group of jobs with similar statistic behaviors. For each simulation run, a predetermined percentage, e.g., 10 percent, of target jobs are created and fed into the cluster at random.

Simulation Settings and Results. In the simulation, the target and background jobs are set at 10 and 90 percent of the total number of jobs, respectively. We evaluate two cases, one with the number of tasks per target job set at one half of the cluster size and the other the same as the cluster size. The tests cover multiple cluster sizes, i.e., 100, 500, 1,000, and 5,000 nodes with each having three replicated servers. All the cases are homogeneous.

The prediction errors for the 99th percentiles of target response times for the two case studies at loads of 50, 75, 80, and 90 percent are shown in Fig. 15. As one can see, the prediction errors are within 15 percent for all the cases studied.

Finally, we note that although the validations for tail latency prediction are exclusively focused on the 99th-percentile tail latency, ForkTail offers similar and consistent performance at higher percentiles, which are not shown here due to the lack of space.

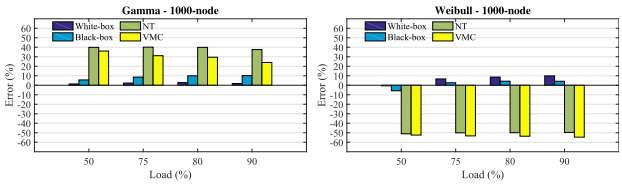


Fig. 17. Comparison of percentage errors in mean latency approximations with M/G/1 queues for Gamma and Weibull service time distributions.

3.2 Mean Latency Prediction Validation

In this section, we extensively validate the predicted mean latencies from ForkMean, for both white-box and black-box approaches, against the results from the existing white-box solutions, the event-driven simulation experiments, and a case study on Amazon EC2 as in Section 3.1.

3.2.1 *Scenario 1: Single-Server Queues*

In this scenario, we compare ForkMean with some well-known closed-form approximations, including NT [10], VMC [36], and VM [37].

Fig. 16 shows the comparison for the systems with 50, 1,000, and 5,000 nodes, each modeled as an M/M/1 queue, at load levels of 50, 75, 80, and 90 percent. Overall, the NT approximation is the most accurate one. The white-box ForkMean yields errors within 5 percent for all the cases studied, which are close to those of the NT approximation. The black-box one that is based on the measured Δ's at 100 node also gives good approximations to mean latency even for the case of 50 nodes, with errors within 10 percent for all the cases. Note that, due to its high computational complexity, the VM approximation is not included in the cases of 1,000 and 5,000 nodes. With small n's, e.g., 50, it is a little better than the VMC approximation but not as good as the NT one.

The NT and VMC approximations above, which are tailored to M/M/1 queues, could not be applied to general service time distributions as the prediction errors are too large to be useful. Indeed, Fig. 17 shows that while both black-box and white-box ForkMean solutions continue to perform well, with errors within 10 percent, VMC and NT offer extremely poor performance with up to 40 and 50 percent errors for Gamma and Weibull task service time distributions, respectively.

The existing methods for the approximation of the mean response time in the case of M/G/1 Fork-Join models are heuristic-based [37] or hybrid-based [13], [29], i.e., combining simulation and analysis. Moreover, these works mainly focus on light-tailed distributions, e.g., Exponential (Exp), Erlang-2 (E2), and Hyperexponential-2 (H2). In contrast, in addition to these distributions, ForkMean solutions are also validated for a wide range of service time distributions.

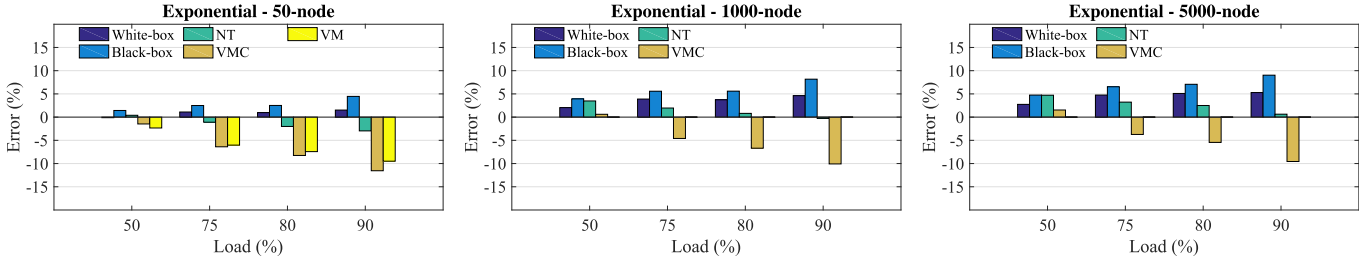


Fig. 16. Comparison of percentage errors in mean latency approximations where each Fork node is modeled as an M/M/1 queuing system.

TABLE 6
Errors for Mean Latency Prediction With M/E2/1 Queues

Load	Method	Number of nodes			
		5	10	15	20
50%	VM	−0.806	−1.486	−1.985	−1.827
	White-box	−7.947	−6.312	−5.483	−4.934
75%	VM	−2.989	−4.587	−5.748	−5.637
	White-box	−9.827	−7.360	−6.316	−5.104
80%	VM	−3.440	−5.336	−6.886	−7.400
	White-box	−10.101	−7.524	−6.666	−5.922
90%	VM	−5.414	−7.885	−9.039	−9.538
	White-box	−11.001	−8.110	−6.398	−5.251

TABLE 7
Errors for Mean Latency Prediction With M/H2/1 Queues

Load	Method	Number of nodes			
		5	10	15	20
50%	VM	−1.007	6.446	13.389	17.937
	White-box	0.869	0.945	1.881	2.118
75%	VM	−1.682	6.556	12.601	16.678
	White-box	−1.255	0.975	2.091	2.574
80%	VM	−0.402	6.361	11.687	14.975
	White-box	−0.106	1.503	2.563	2.753
90%	VM	0.111	4.030	6.366	8.697
	White-box	−0.081	1.183	1.242	1.825

To test the effectiveness of ForkMean, we first compare our white-box solution with the heuristic approximations in [37] for the cases of Erlang-2 (E2) and Hyperexponential-2 (H2) service time distributions with Poisson arrivals, i.e., M/G/1 queues.

Tables 6 and 7 present the comparative results for Erlang-2 and Hyperexponential-2, respectively. Again due to the computational complexity concerning the VM approximation, we perform comparison only for small n's, i.e., up to 20, the maximum problem size studied by the authors of the VM approximation [37], although our solution offers consistent performance at large n's as well. For the Erlang-2 distribution, the VM approach gives better predictions at load level of 50 percent and lower numbers of nodes, i.e., 5 and 10 nodes, while our solution yields comparable or better predictions for the other settings. The accuracy of our approach outperforms that of the VM for the Hyperexponential distribution. Although yielding good prediction performance for systems with small numbers of Fork nodes, the VM approximation faces the issue of numerical instatibility and computational complexity due to big binomial coefficients, resulting in higher prediction errors for higher numbers of nodes, as observered from the reported results. In additon, while the VM approximation can in theory be applied to G/G/1 queues, finding light and heavy traffic limits for an arbitrary service time distribution, e.g., Weibull or truncated Pareto, is nontrivial.

Fig. 18 shows the prediction accuracy of ForkMean for the above heavy-tailed service time distributions. Both white-box and black-box solutions yield quite accurate predictions for less heavy-tailed distributions, i.e., Weibull, for all the cases studied, with errors within 12 percent for all the cases. For heavier tailed distributions, i.e., truncated Pareto and empirical, the solutions give good approximations at high load levels, i.e., 80 percent or higher, a region of interest for resource provisioning. Overall, the black-box solution gives comparably close prediction performance to that of the white-box one. The errors are mostly within 20 and 10 percent at the load levels of 80 and 90 percent, respectively.

The predictions for G/G/1 cases as in Section 3.1 also show similar performance, i.e., within 20 percent errors at the load levels of 80 percent or higher, which are not shown here due to the lack of space.

3.2.2 *Scenario 2: Systems With Replicated Servers*

We now validate ForkMean for systems with 3-replica Fork nodes. We consider two dispatching policies, i.e., Round-Robin and redundant-task-issue, and heavy-tailed service time distributions as in Section 3.1. The validation is run only for the black-box solution since the exact service time distributions for the Fork nodes are simply unknown for such cases.

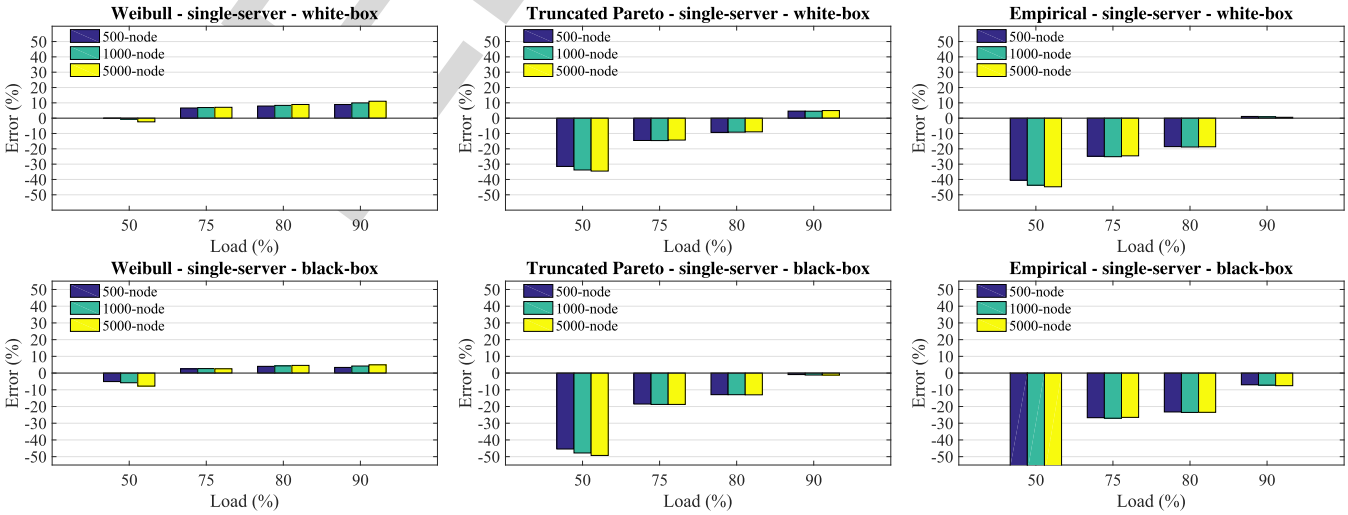


Fig. 18. Errors for mean response time approximations using the white-box (upper row) and black-box (lower row) solutions.

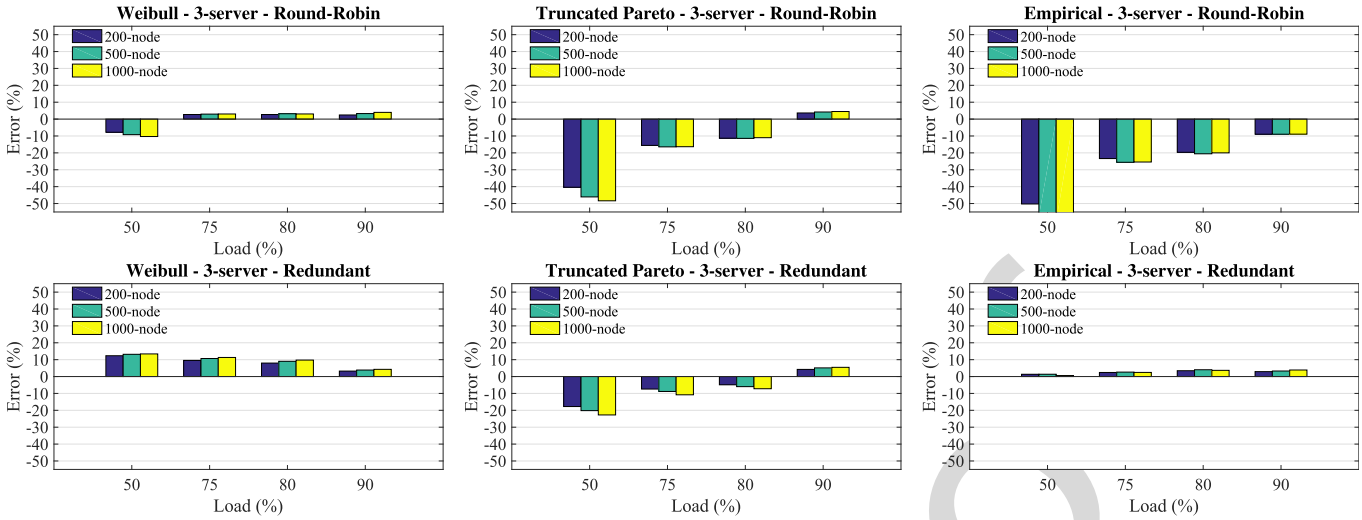


Fig. 19. Errors in mean response time approximation for systems with replicated servers applying Round-Robin (upper row) and redundant-task-issue (lower row) policies.

Fig. 19 presents the results for these cases using the black-box approach, applying the Δ values measured from the respective systems at $n = 100$ to the ones with 200, 500, and 1,000 nodes. One can see that the results for the Round-Robin cases are close to those in the previous scenario. This is due to the fact that the Round-Robin policy mainly performs load balancing between replica and thus the effective service time distributions on the Fork nodes are almost unchanged. In contrast, the model yields good predictions for the redundant-task-issue policy for the entire load range under study. This is largely because this policy curbs the tail effects and makes the effective service time distributions less heavy-tailed. These results agree with those from the previous scenarios for less heavy-tailed distributions, i.e., Gamma and Weibull.

3.2.3 Scenario 3: Systems With Variable Numbers of Tasks

For illustrative purposes, we validate the results on Fork-Join models with homogeneous, single-server Fork nodes with the above service time distributions using the black-box solution, assuming that the tasks for each incoming job is randomly dispatched to 40–60 percent total number of Fork nodes. As a result, the effective load on each Fork node is half of that on the single-server systems in Scenario 3.2.1. Therefore, we double the arrival rate, λ, to keep the same arrival rate on each node as in the previous cases. The results of this scenario are shown in Fig. 20. Similar to the previous scenarios, the black-box solution gives accurate predictions across the entire load range for light-tailed distributions, e.g., Exponential, Gamma (which is not shown here), while yielding good approximations for the heavy-tailed distributions, i.e., truncated Pareto and empirical, at high load regions, e.g., 80 percent or above.

3.2.4 Scenario 4: A Case Study on Amazon EC2

We also evaluate the accuracy of the black-box solution for the case study on AWS EC2 as in Section 3.1.1. To illustrate the effectiveness of the black-box solution for this case study, we compute the gap for the 32-worker cluster and apply it to the approximation of request mean response time for the case of the 64-worker cluster. Table 8 presents the prediction errors for this case study. Again, the black-box method predicts mean response time quite accurately when the system at the effective load of 60 percent or higher, corresponding to arrival rates greater than 3.5 requests/s.

Finally, we note that the tail effect is a recognized issue in datacenter applications and tail-cutting techniques are often exploited in datacenters to reduce the tail effects [1], [15], [16], [38]. As a result, the effective service time distributions tend to be less heavy-tailed. Therefore, ForkTail and Fork-Mean show a great potential to be able to accurately predict the tail and mean latencies in a wide load range in practice, not limited to a high load region.

4 Sensitivity Analysis

From all the experiments above, we can see that the proposed approximations can be applied to a wide range of

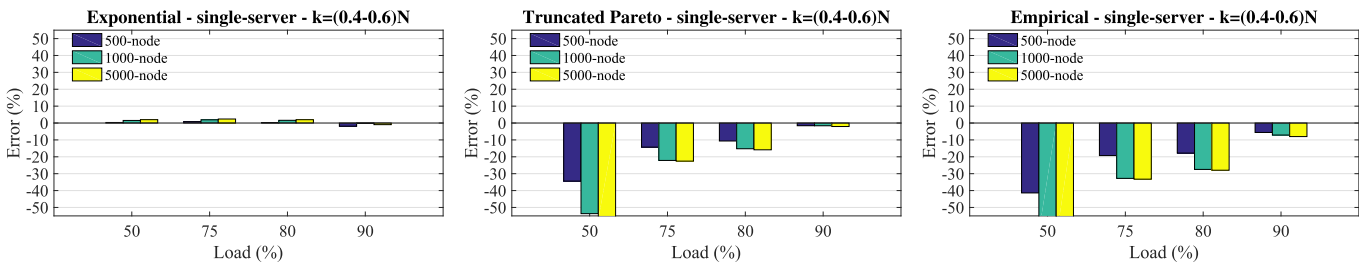


Fig. 20. Errors in mean response time approximation for systems with variable numbers of tasks.

TABLE 8
Errors in Mean Response Time Approximation Using the Black-Box Solution for the Test Case on AWS

	Effective load (Arrival rate (requests/s))					
	50.0%	58.4%	66.7%	75.1%	83.4%	91.7%
#workers	**(3.0)**	**(3.5)**	**(4.0)**	**(4.5)**	**(5.0)**	**(5.5)**
64	31.678	10.489	7.817	8.874	15.274	13.991

systems with reasonable prediction errors for the 99th percentile and mean job latency, consistently within 20 and 15 percent at the loads of 80 and 90 percent, respectively. Now, the question yet to be answered is how much impact these errors will have on the accuracy for resource provisioning at high loads. To this end, we conduct a sensitivity analysis of tail and mean latencies as functions of load.

We perform experiments with different load levels in the high load region, i.e., 78 to 95 percent, for FJQNs with different service time distributions, i.e., exponential, Weibull, truncated Pareto, and empirical ones. Figs. 21 and 22 shows results from both simulation and the proposed approximations for 1,000-node systems. First, we note that the proposed models consistently overestimates the tail and mean latencies for the exponential and Weibull cases, while mostly underestimates them for the truncated Pareto and empirical cases. In other words, the former causes resource overprovisioning, whereas the latter leads to resource underprovisioning. Then the question is how much. Take the exponential case as an example, the predicted mean latency at 90 percent load is roughly equal to the simulated one at 91 percent load. This means that the model may lead to 1 percent resource over provisioning for the exponential cases. Following the same logic, it is easy to find that for both exponential and Weibull cases, the prediction models for both tail and mean latency may result in no more than 1 percent resource overprovisioning in the entire 78–95 percent load range. By the same token, we find that for the truncated Pareto and empirical cases, the models may cause up to 4 and 6 percent resource underprovisioning at 80 percent load and 2 and 1 at 90 percent load for tail and mean latency, respectively. This can be well compensated for by leaving a 6 percent resource margin in practice. This implies that in the worst-case when the actual service time distribution is light-tailed, our approximations may cause up to 7 percent resource overprovisioning at the loads of 80 percent or higher, given that we don't have the knowledge about the tail-heaviness of the workload. With the prediction and the small overprovisioning to compensate the prediction error proposed in this paper, one can expect to run the system at up to 90 percent instead of 50 percent resource utilization with tail and mean latency guarantee.

Our sensitivity analyses for the other Fork-Join structures, which are not shown here, have led to similar conclusions. This demonstrates the effectiveness of our prediction models as a powerful means to facilitate multi-SLO-guaranteed, e.g., tail and mean latency guaranteed job scheduling and resource provisioning for datacenter applications.

5 Applicability Range

In this section, we want to answer the following question: In what parameter range can our models predict the request latency within 20 percent errors at high load? To this end, we note that we need to focus on identifying the applicability range on the heavy tail end, rather than the light tail end for two reasons. First, from the extensive experiments above, we found that our methods give quite accurate approximations for tail and mean latency for a wide range of loads for light-tailed distributions, e.g., Exponential, Gamma, and Erlang-2. Second, in practice, server wokloads in datacenters exhibit heavy-tailed distributions [15], [32]. Also, the heavy-tailed truncated Pareto distribution given in Eq. (13) was found to be a good fit for empirical data from server workloads [31]. Hence, in what follows, we test the applicability range of our approximations based on this distribution.

From extensive experiments with the truncated Pareto distribution, we found that our approximations predict the tail and mean latencies within 20 percent errors at the loads of 80 percent or higher, when the tail index α in Eq. (13) is less than 2, i.e., $0 < \alpha < 2$. This range of α was found to be large enough to cover the server workloads in [31].

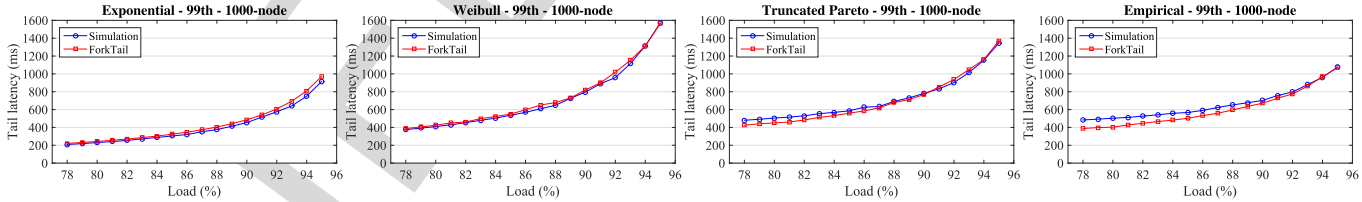


Fig. 21. Differences in the 99th percentile response times from simulation and ForkTail for 1000-node systems with different service time distributions and fixed number of Fork tasks.

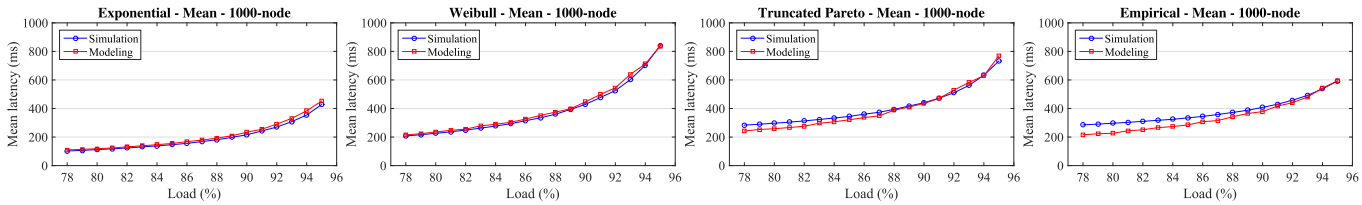


Fig. 22. Differences in mean response times from the simulation and black-box ForkMean for 1000-node systems with different service time distributions and fixed number of Fork tasks.

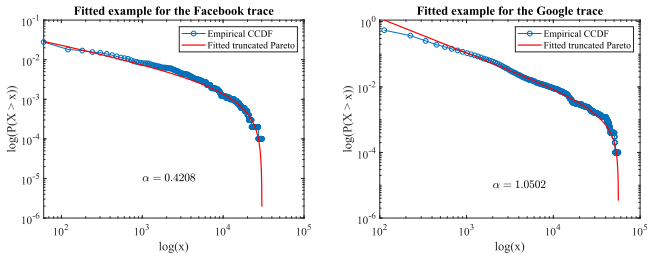


Fig. 23. Examples of fitting the truncated Pareto distribution to sampled data from Facebook and Google traces. The plots show the complementary CDF (CCDF), which is on a log scale, to focus on the tail portion of the distribution.

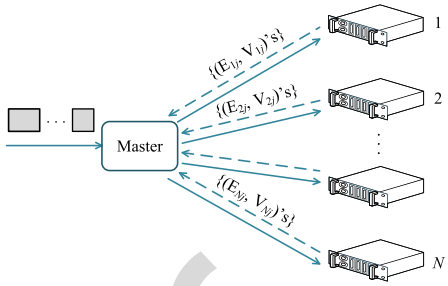


Fig. 24. A hybrid, centralized-and-distributed job scheduler.

To further test if today's datacenter workloads indeed fall into the above range, we test the fitting of the truncated Pareto distribution to the workload traces from Facebook and Google provided in [19]. These traces include a mixture of different types of workloads placed on datacenter servers. To simulate the workload on one server, we draw 10,000 random samples from each trace and fit them to the truncated Pareto distribution based on the procedure suggested in [39], which uses the $(r+1)$ largest-order statistics and visual check. We found that the fitted values of α for Google and Facebook samples are mostly within the applicability range of $(0, 2)$. Fig. 23 illustrates two examples of the fitted curves.

The above results strongly suggest that our proposed methods can indeed serve as a useful tool for the approximation of tail and mean latency for datacenter workloads.

6 Facilitating Job Scheduling and Resource Provisioning

We now discuss how our proposed approximations may be used to facilitate both SLO-guaranteed job scheduling and resource provisioning. We present here only the procedures for tail latency approximation, i.e., ForkTail. The procedures for mean latency follow similar steps since the approximation of mean latency is based on ForkTail. The proposed ideas are preliminary and somewhat sketchy, but yet, they do help reveal the promising prospects of our proposed model and point directions for future studies on this topic.

Job Scheduling. We describe the ideas of how a tail-latency-SLO-guaranteed hybrid centralized-and-distributed job scheduler can be developed, based on ForkTail. The main idea is to rely on distributed measurement of the means and variances of the task response times and centralized decision making as to how and whether the request tail-latency SLO can be met, as depicted in Fig. 24. In the master server on the left resides the central job scheduler to which users submit their requests with given tail-latency SLOs. All the servers in the cluster measures the means and variances of task response times for tasks of different sizes or in different bins on a continuous basis. All the servers periodically convey their measurements to the central scheduler. Upon the arrival of a request with a given tail-latency SLO and given k tasks to spawn, based on Eq. (5), the central scheduler will run a Fork-node selection algorithm to determine which k Fork nodes should be used such that the tail-latency SLO can be met. If such k Fork nodes are found, the request will be admitted, otherwise, either the tail-latency SLO will be renegotiated or the request will be rejected. At runtime, the central scheduler periodically run the prediction model using the up-to-date means and variances as input to ensure that the tail-latency SLOs for the on-going requests continue to be met.

Resource Provisioning. ForkTail for the homogeneous case (i.e., Eqs. (8) and (9)) naturally enables a resource provisioning solution involving two steps: (a) the evaluation of the task-level performance requirements to achieve a given tail-latency SLO; and (b) the selection of an underlying platform to meet the requirements. Here, step (a) is platform independent and hence is portable to any datacenter platforms.

For example, consider a service deployment scenario with a given tail-latency SLO and a minimum throughput requirement, R. Assuming that N, k_i, and $P(K = k_i)$ for the given service are known, Eq. (9) can be used to first translate the tail-latency SLO into a pair, i.e., the mean and variance of the task response time. This pair then serves as the task performance budgets or the task-level performance requirements, which are platform independent and portable. This completes step (a).

In step (b), a Fork node is set up, e.g., using three virtual machine instances purchased from Amazon EC2 to form a 3-replica Fork node, loaded with a data shard in the memory. Then run tasks at increasing task arrival rate λ until the measured task mean and/or variance are about to exceed the corresponding budget(s). At this arrival rate λ, the tail-latency SLO is met without resource over-provisioning. In other words, the λ value at this point would be the maximum sustainable task throughput, or equivalently, the request throughput, in order to meet the tail-latency SLO. If this throughput is greater than R, the minimum throughput requirement is also met. This means that the resource provisioning is successful and a cluster with $3N$ VM instances can be deployed. Otherwise, repeat step (b) by using a more powerful VM instance or with a re-negotiated tail-latency SLO and/or minimum throughput requirement.

7 Related Work

Fork-Join structures are traditionally modeled by FJQNs, which have been studied extensively in the literature. To date, the exact solution exists for a two-Fork-node FJQN only [10], [40]. Most of the previous works primarily focus on the approximation of mean response time [10], [11], [41] and its bounds [42], [43]. For networks with general service time distribution, several works have introduced hybrid approaches that combine analysis and simulation to derive the empirical approximation for mean response time [10], [13].

Some analytic results are available on redundant task issues [44], [45], [46]. They either address only a single replicated

server subsystem with exponential task service time distribution [45] or parallel request load balancing without task spawning [44], [46].

Tail Latency Approximation. In terms of tail-latency related research, several works dealt with the approximation of response time distribution assuming a simple queuing model for each Fork node, e.g., M/M/1 [47] or M/M/k [12]. Computable stochastic bounds on request waiting and response time distributions for some FJQNs are provided in a recent work [48]. The most interesting and relevant work is given in [14]. The authors of this work proposed a method, called EAT, for the approximation of tail latency for homogeneous FJQNs based on the analytical results from single-node and two-node cases. The approximation applies to FJQNs with any service time distribution that can be transformed into a phase-type distribution. Although outperforming our solutions by a few percentage points in terms of tail prediction, its computational complexity renders it infeasible to facilitate online resource provisioning. Moreover, this work can only cover a small fraction of the aforementioned design space and hence, cannot be used to facilitate resource provisioning in practice.

Mean Latency Approximation. Various works have been proposed for the approximation of mean response time of FJQNs using model-based or hybrid approaches. The work in [10] introduces a hybrid approach for the approximation of mean response time, R_n, for a Fork-Join model with n M/M/1 Fork nodes ($2 \leq n \leq 32$) based on the exact solution for the 2-way network [40] and simulation. In [36], the authors proposed an approximation for mean response time based on the optimistic and pessimistic bounds. Another approximation for mean response time of Fork-Join models with general inter-arrival and service time distributions is proposed in [37] based on light traffic interpolation and heavy traffic limit. The light traffic interpolation is computed from the mean response time of the Fork-Join network when there is only a tagged job in the network, which is equivalent to the maximum of task service time random variables. The heavy traffic limit is postulated based on the observation of the relationship between expressions for light and heavy traffic for 1-way and 2-way networks. In [29], the authors proposed a hybrid procedure for the approximation of mean response time for Fork-Join models with M/G/1 queues. Indeed, this work proposed a methodology rather than specific expressions for finding mean response time. In a recent work [49], a simulation study assessed the accuracy of the approximation based on order statistic.

The existing approaches above are white-box solutions targeting at individual Fork-Join models with specific queuing server models. In contrast, in this paper, we propose both white-box and black-box solutions, applicable to Fork-Join networks with arbitrary server models.

SLO-Aware Resource Provisioning. Due to the lack of theoretical underpinning, the existing SLO-aware resource provisioning proposals cannot provide tail and/or mean latencies SLO guarantee by design. Instead, various techniques such as tail-cutting techniques [15], [16], a combination of job priority and rate limiting based on network calculus [50] are employed to indirectly provide high assurance of meeting tail-latency SLOs. As indirect solutions, however, they cannot ensure precise resource allocation to meet tail-latency SLOs, while allowing high resource utilization, and hence may result in resource overprovisioning. Yet, another alternative solution is to track the target tail-latency SLO through online, direct tail-latency measurement and dynamic resource provisioning [51], [52]. This approach, however, may not be effective, especially in enforcing stringent tail latency SLOs. To see why this is true, consider the 99.9th percentile request response time of 200 ms, i.e., probabilistically, only one out of 1,000 requests should experience a response time greater than 200 ms. Assume that the average request arrival rate is 50 per second. To track, through direct tail-latency measurement, whether this tail latency SLO is violated or not with reasonably high confidence, one needs to collect, e.g., 100K samples to see if there are more than 100 requests whose response times exceed 200 ms. This, however, takes about 100K/50 = 2000 seconds or about 33 minutes of measurement time! Given possibly high volatility of datacenter workloads, the tail latency SLO may have been violated multiple times during this measurement period, even though the total number of requests whose response times exceeding 200 ms may be well within 100. In constrast, using our proposed models, with only 20 seconds of measurement time, one can collect $20 \times 50 = 1000$ task samples at individual Fork nodes to allow a reasonably accurate estimation of the means and variances of task response times. With moving average for a given time window, e.g., 20 seconds, these means and variances and hence, the 99.9th percentile, can be updated every tens of milliseconds, making it possible to enable fast online tail-latency-guaranteed job scheduling and resource provisioning.

In summary, a solution that can predict the tail and/or mean latency using a small number of samples collected in a short period of time as input and that applies to a large design space of Fork-Join structures must be sought, the primary motivation of the current work.

8 Conclusion and Future Work

A key challenge in enabling tail-latency and/or mean-latency SLOs for data-intensive services and applications in datacenters is how to predict the latencies for a broad range of Fork-Join structures underlying those services and applications. In this paper, we proposed to study a generic black-box Fork-Join model for the approximations of tail and mean latency that covers most Fork-Join structures of practical interests. On the basis of a central limit theorem for queuing servers under heavy load, we were able to arrive at approximate solutions to this model for both tail and mean latencies, called ForkTail and ForkMean, respectively. These approximations were found to be able to predict the tail and mean latencies for most practical scenarios consistently within 20 percent in a load region of 80 percent or higher, resulting in at most 7 percent resource overprovisioning, making it a powerful tool for resource provisioning at high load. Finally, we discussed some preliminary ideas of how to make use of the proposed prediction model to facilitate tail-latency-SLO-guaranteed job scheduling and resource provisioning.

In our future work, based on ForkTail and ForkMean, we shall develop both job scheduling and online/offline resource provisioning solutions with tail-latency and/or mean-latency SLO guarantee.

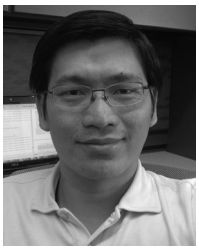
ACKNOWLEDGMENTS

This work is supported by the NSF under awards CCF XPS 1629625 and CCF 1704504.

REFERENCES

[1] M. Jeon *et al.*, "Predictive parallelization: Taming tail latencies in web search," in *Proc. 37th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2014, pp. 253–262.
[2] J. Brutlag, "Speed matters for Google web search," 2009. [Online]. Available: https://services.google.com/fh/files/blogs/google_delayexp.pdf
[3] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Conf. Symp. Operating Syst. Des. Implementation*, 2004, pp. 137–150.
[4] Apache spark, Accessed: Feb. 26, 2020. [Online]. Available: https://spark.apache.org
[5] G. Blake and A. G. Saidi, "Where does the time go? Characterizing tail latency in memcached," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2015, pp. 21–31.
[6] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale," in *Proc. 3rd ACM Symp. Cloud Comput.*, 2012, pp. 7:1–7:13.
[7] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware cluster management," in *Proc. 19th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2014, pp. 127–144.
[8] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation With Computer Science Applications*. Hoboken, NJ, USA: Wiley-Interscience, 2006.
[9] A. Thomasian, "Analysis of fork/join and related queueing systems," *ACM Comput. Surv.*, vol. 47, no. 2, pp. 1–71, 2014.
[10] R. Nelson and A. N. Tantawi, "Approximate analysis of fork/join synchronization in parallel queues," *IEEE Trans. Comput.*, vol. 37, no. 6, pp. 739–743, Jun. 1988.
[11] E. Varki, "Response time analysis of parallel computer and storage systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 11, pp. 1146–1161, Nov. 2001.
[12] S. S. Ko and R. F. Serfozo, "Response times in M/M/s fork-join networks," *Advances Appl. Probability*, vol. 36, no. 3, pp. 854–871, 2004.
[13] R. J. Chen, "A hybrid solution of fork/join synchronization in parallel queues," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 8, pp. 829–845, Aug. 2001.
[14] Z. Qiu, J. F. Pérez, and P. G. Harrison, "Beyond the mean in fork-join queues: Efficient approximation for response-time tails," *Perform. Eval.*, vol. 91, pp. 99–116, 2015.
[15] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013.
[16] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *Proc. 9th ACM Conf. Emerg. Netw. Experiments Technol.*, 2013, pp. 283–294.
[17] N. J. Yadwadkar, G. Ananthanarayanan, and R. Katz, "Wrangler: Predictable and faster jobs using fewer resources," in *Proc. ACM Symp. Cloud Comput.*, 2014, pp. 26:1–26:14.
[18] R. Nishtala *et al.*, "Scaling memcache at Facebook," in *Proc. 10th USENIX Conf. Netw. Syst. Des. Implementation*, 2013, pp. 385–398.
[19] P. Delgado, F. Dinu, A. M. Kermarrec, and W. Zwaenepoel, "Hawk: Hybrid datacenter scheduling," in *Proc. USENIX Conf. Usenix Annu. Tech. Conf.*, 2015, pp. 499–510.
[20] J. F. C. Kingman and M. F. Atiyah, "The single server queue in heavy traffic," *Proc. Cambridge Philosophical Soc.*, vol. 57, pp. 902–904, 1961.
[21] J. Köllerström, "Heavy traffic theory for queues with several servers. I," *J. Appl. Probability*, vol. 11, no. 3, pp. 544–552, 1974.
[22] M. Nguyen, Z. Li, F. Duan, H. Che, Y. Lei, and H. Jiang, "The Tail at Scale: How to Predict It?" in *Proc. 8th USENIX Workshop Hot Topics Cloud Comput.*, 2016.
[23] S. Sani and O. A. Daman, "Mathematical modeling in heavy traffic queuing systems," *Amer. J. Operations Res.*, vol. 4, pp. 340–350, 2014.
[24] R. D. Gupta and D. Kundu, "Generalized exponential distributions," *Australian New Zealand J. Statist.*, vol. 41, no. 2, pp. 173–188, 1999.
[25] M. Nguyen, S. Alesawi, N. Li, H. Che, and H. Jiang, "ForkTail: A black-box fork-join tail latency prediction model for user-facing datacenter workloads," in *Proc. 27th Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2018, pp. 206–217.
[26] L. Kleinrock, *Queueing Systems*, Vol. 1: Theory. Hoboken, New Jersey, USA: Wiley, 1975.
[27] W. Whitt, "The queueing network analyzer," *The Bell Syst. Tech. J.*, vol. 62, no. 9, pp. 2779–2815, Nov. 1983.
[28] G. Brys, M. Hubert, and A. Struyf, "Robust measures of tail weight," *Comput. Statist. Data Anal.*, vol. 50, no. 3, pp. 733–759, 2006.
[29] A. Thomasian and A. N. Tantawi, "Approximate solutions for M/G/1 fork/join synchronization," in *Proc. 26th Conf. Winter Simul.*, 1994, pp. 361–368.
[30] L. A. Barroso, J. Dean, and U. Hölzle, "Web search for a planet: The Google cluster architecture," *IEEE Micro*, vol. 23, no. 2, pp. 22–28, 2003.
[31] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*, 1st ed. Cambridge, U.K.: Cambridge Univ. Press, 2013.
[32] D. Meisner, W. Junjie, and T. F. Wenisch, "BigHouse: A simulation infrastructure for data center systems," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2012, pp. 35–45.
[33] Apache hadoop, Accessed: Feb. 26, 2020. [Online]. Available: https://hadoop.apache.org
[34] Y. Chen, S. Alspaugh, and R. Katz, "Interactive analytical processing in big data systems: A cross-industry study of MapReduce workloads," *Proc. VLDB Endowment*, vol. 5, no. 12, pp. 1802–1813, Aug. 2012.
[35] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *Proc. 5th Eur. Conf. Comput. Syst.*, 2010, pp. 265–278.
[36] E. Varki, A. Merchant, and H. Chen, "The M/M/1 fork-join queue with variable sub-tasks," 2002. [Online]. Available: http://www.cs.unh.edu/ varki/publication/2002-nov-open.pdf
[37] S. Varma and A. M. Makowski, "Interpolation approximations for symmetric fork-join queues," *Perform. Eval.*, vol. 20, no. 1/3, pp. 245–265, 1994.
[38] L. Suresh, M. Canini, S. Schmid, and A. Feldmann, "C3: Cutting tail latency in cloud data stores via adaptive replica selection," in *Proc. 12th USENIX Conf. Netw. Syst. Des. Implementation*, 2015, pp. 513–527.
[39] I. Aban, M. Meerschaert, and A. Panorska, "Parameter estimation for the truncated pareto distribution," *J. Amer. Statist. Assoc.*, vol. 101, no. 473, pp. 270–277, 2006.
[40] L. Flatto and S. Hahn, "Two parallel queues created by arrivals with two demands I," *SIAM J. Appl. Math.*, vol. 44, no. 5, pp. 1041–1053, 1984.
[41] F. Alomari and D. A. Menasce, "Efficient response time approximations for multiclass fork and join queues in open and closed queuing networks," *IEEE Trans. Parallel Distributed Syst.*, vol. 25, no. 6, pp. 1437–1446, Jun. 2014.
[42] S. Balsamo, L. Donatiello, and N. M. Van Dijk, "Bound performance models of heterogeneous parallel processing systems," *IEEE Trans. Parallel Distributed Syst.*, vol. 9, no. 10, pp. 1041–1056, Oct. 1998.
[43] R. J. Chen, "An upper bound solution for homogeneous fork/join queuing systems," *IEEE Trans. Parallel Distributed Syst.*, vol. 22, no. 5, pp. 874–878, May 2011.
[44] D. Wang, G. Joshi, and G. Wornell, "Efficient task replication for fast response times in parallel computation," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 1, pp. 599–600, Jun. 2014.
[45] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytiä, and A. Scheller-Wolf, "Reducing latency via redundant requests: Exact analysis," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 2015, pp. 347–360.
[46] Z. Qiu and J. F. Perez, "Evaluating the effectiveness of replication for tail-tolerance," in *Proc. 15th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2015, pp. 443–452.
[47] S. Balsamo and I. Mura, "Approximate response time distribution in Fork and Join systems," in *Proc. ACM SIGMETRICS Joint Int. Conf. Meas. Model. Comput. Syst.*, 1995, pp. 305–306.
[48] A. Rizk, F. Poloczek, and F. Ciucu, "Computable bounds in fork-join queueing systems," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 2015, pp. 335–346.
[49] A. Lebrecht and W. J. Knottenbelt, "Response time approximations in fork-join queues," in *Proc. 23rd Annu. UK Perform. Eng. Workshop*, 2007.
[50] T. Zhu, A. Tumanov, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger, "PriorityMeister: Tail latency QoS for shared networked storage," in *Proc. ACM Symp. Cloud Comput.*, 2014, pp. 29:1–29:14.

[51] A. Wang, S. Venkataraman, S. Alspaugh, R. Katz, and I. Stoica, "Cake: Enabling high-level SLOs on shared storage systems," in *Proc. 3rd ACM Symp. Cloud Comput.*, 2012, pp. 14:1–14:14.
[52] A. D. Ferguson, P. Bodik, E. Boutin, and R. Fonseca, "Jockey: Guaranteed job latency in data parallel clusters," in *Proc. 7th ACM Eur. Conf. Comput. Syst.*, 2012, pp. 99–112.



Minh Nguyen received the BS and MS degrees in electrical engineering from the Ho Chi Minh City University of Technology, Vietnam; and the PhD degree in computer engineering from the University of Texas at Arlington, Arlington, Texas. He is currently a lead hardware integration engineer at Ikon Technologies. His current research interests include datacenter resource management and job scheduling, edge computing, IoT, and smart cities.

Sami Alesawi received the BS degree in computer engineering and the MS degree in computer science from King Abdulaziz University, Jeddah, Saudi Arabia, and the PhD degree from The University of Texas at Arlington, Arlington, Texas. He is currently working as an assistant professor at the Faculty of Computing and Information Technology in Rabigh, King Abdulaziz University, Saudi Arabia. His current research interests include datacenter resource management and job scheduling.

Ning Li received the BSc degree in computer science from Jiangsu University, China; the MSc degree in computer engineering from the Nanjing University of Science and Technology, China; and the PhD degree in computer system architecture from the Huazhong University of Science and Technology, China. He is currently working as a post-doc research associate with the University of Texas at Arlington, Arlington, Texas. His research interests include virtualization, quality of service, cloud computing and storage systems.

Hao Che (Senior Member, IEEE) received the BS degree from Nanjing University, Nanjing, China; the MS degree in physics from the University of Texas at Arlington, Arlington, Texas; and the PhD degree in electrical engineering from the University of Texas at Austin, Austin, Texas. He is currently a full professor in the Department of Computer Science and Engineering, University of Texas at Arlington, Texas. Prior to joining UTA, he was a system architect with Santera Systems, Inc. in Plano (2000–2002) and an assistant professor of electrical engineering at the Pennsylvania State University (1998 to 2000). His current research interests include network architecture and Internet traffic control, datacenter resource management and job scheduling, edge computing and IoT.

Hong Jiang (Fellow, IEEE) received the BSc degree in computer engineering from the Huazhong University of Science and Technology, Wuhan, China; the MASc degree in computer engineering from the University of Toronto, Toronto, Canada; and the PhD degree in computer science from the Texas A&M University, College Station, Texas. He is currently chair and Wendell H. Nedderman Endowed professor of Computer Science and Engineering Department, University of Texas at Arlington, Arlington, Texas. Prior to joining UTA, he served as a program director at National Science Foundation (2013–2015) and he was at University of Nebraska-Lincoln since 1991, where he was Willa Cather professor of Computer Science and Engineering. He has graduated 17 PhD students and supervised 20 post-doctoral fellows and visiting scholars. He is currently supervising/co-supervising more than 10 PhD students and post-doc fellows. His present research interests include computer architecture, computer storage systems and parallel I/O, high-performance computing, big data computing, and cloud and edge computing. He is an associate editor of the *IEEE Transactions on Computers* and recently served as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems*. He has more than 300 publications in major journals and international Conferences in these areas, including the *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *Proceedings of IEEE*, *ACM Transactions on Architecture and Code Optimization*, the *ACM Transactions on Storage*, USENIX ATC, FAST, EUROSYS, ISCA, MICRO, SOCC, LISA, SIGMETRICS, ICDE, DATE, ICDCS, IPDPS, MIDDLEWARE, OOPLAS, ECOOP, SC, ICS, HPDC, INFOCOM, ICPP, etc., and his research has been supported by NSF and industry. He is a member of the ACM.