ForkTail: A Black-Box Fork-Join Tail Latency Prediction Model for User-Facing Datacenter Workloads

Minh Nguyen
The University of Texas at Arlington
mqnguyen@mavs.uta.edu

Sami Alesawi*
The University of Texas at Arlington
salesawi@kau.edu.sa

Ning Li
The University of Texas at Arlington
ning.li@uta.edu

Hao Che
The University of Texas at Arlington
hche@cse.uta.edu

Hong Jiang
The University of Texas at Arlington
hong.jiang@uta.edu

ABSTRACT

The workflows of the predominant user-facing datacenter services, including web searching and social networking, are underlaid by various Fork-Join structures. Due to the lack of understanding the performance of Fork-Join structures in general, today's datacenters often resort to resource overprovisioning, operating under low resource utilization, to meet stringent tail-latency service level objectives (SLOs) for such services. Hence, to achieve high resource utilization, while meeting stringent tail-latency SLOs, it is of paramount importance to be able to accurately predict the tail latency for a broad range of Fork-Join structures of practical interests.

In this paper, we propose ForkTail, a black-box Fork-Join tail latency prediction model that covers a wide range of Fork-Join structures. In ForkTail, all Fork nodes are treated as black boxes, admitting both homogeneous and inhomogeneous cases, and different requests in the request flow are allowed to spawn different numbers of tasks forked to different numbers of Fork nodes. On the basis of the central limit theorem for queuing models under heavy load, we are able to arrive at a highly computational effective, empirical expression for the tail latency as a function of the means and variances of the task response times. Since this expression can be applied to request sub-flows at any granularities, it can be used for tail-latency prediction for services in a consolidated environment, where different services and applications may share the same datacenter cluster resources. 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 expression can consistently predict the tail latency within 20% and 15% prediction errors at 80% and 90% load levels, respectively. Moreover, our sensitivity analysis demonstrates that such errors can be well compensated for with no more than 5% and 3% resource overprovisioning at these two load levels, respectively. This, together with its extremely low computational complexity, makes ForkTail a viable tool for both offline and online job scheduling and resource provisioning for user-facing datacenter applications.

CCS CONCEPTS

• **Computing methodologies** → **Modeling methodologies**;

KEYWORDS

tail latency, Fork Join queuing networks, datacenter resource provisioning, user-facing datacenter workloads.

1 INTRODUCTION

Fork-Join structures underlay many user-facing datacenter services, including web searching and social networking. A Fork-Join structure is a critical building block in the request processing workflow that constitutes a major part of request processing time and hardware cost, e.g., more than two-third of the total processing time and 90% hardware cost for a Web search engine [22]. In a Fork-Join structure (e.g., Fig. 1), each request in a flow spawns multiple tasks, which are forked to, queued and processed at different fork nodes in parallel; the task results are merged at a join node; and the final results are returned. Due to barrier synchronization, the request response time is determined by the slowest task, making it extremely challenging to predict the request performance, in particular, the request tail latency. This is because tail latency is a probabilistic performance measure concerning the tail probability, which is hard to capture, from both modeling and measurement points of view. In particular, it is harder but more important[1] to predict the tail 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 1ms at the load of 75% to 1s at the load of 89% [8].

Tail latency is considered to be the most important performance measure for user-facing datacenter applications [10] and it is normally expressed as a high percentile request response time, e.g., the 99th percentile response time of 200ms, to satisfy as many user requests as possible. Unfortunately, without a good tail-latency prediction model, especially in the high load region, to provide high assurance of meeting tail-latency SLOs for user-facing services, the current practice is to overprovision resources, which however, results in low resource utilization in datacenters [16, 34]. For example,

*The author is also with Faculty of Computing and Information Technology in Rabigh, King Abdulaziz University, Saudi Arabia

HPDC '18, June 11–15, 2018, Tempe, AZ, USA

ACM ISBN 978-1-4503-5785-2/18/06…$15.00
https://doi.org/10.1145/3208040.3208058

[1] In the low load region, tail-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 tail-latency requirements.

aggregate CPU and memory utilizations in a 12,000-server Google cluster are mostly less than 50%, leaving 50% and 40% allocated CPU and memory resources, respectively, idle almost at any time [34]. Similarly, in a large production cluster at Twitter, aggregate CPU usage is within 20%–30% even thought CPU reservations are up to 80% and aggregate memory usage is mostly within 40%–50% while memory allocation consistently exceeds 75% [16]. Hence, how to improve resource utilization or the load from currently less than 50% to, say, 80-90%, while meeting stringent SLOs has been a challenging issue for datacenter service providers [16]. To this end, *a key challenge to be tackled is how to accurately capture the tail latency with respect to various Fork-Join structures at high load.*

Fork-Join structures are traditionally modeled by a class of queuing network models, known as Fork-Join queuing networks (FJQNs) [9]. FJQNs are white-box models in the sense that all the Fork nodes are explicitly modeled as queuing servers with given 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 queuing servers are M/M/1 queues [37]. Although empirical solutions for some FJQNs are available, e.g., [11, 25, 29, 33, 38], 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) [9]; (b) the case with multiple replicated servers per Fork node for failure recovery, task load balancing, and/or redundant task issues for tail cutting [14, 39] or fast recovery from straggling tasks [42]; (c) the case where the number of spawned tasks per request may vary from one request to another [31]; and (d) the case of consolidated services, where different types of services and applications may share the same datacenter cluster resources [15]. 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, called ForkTail, 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. It also allows the number of spawned tasks per request, 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 solution to this black-box model indeed adequately covers the above design space.

However, a general solution to ForkTail is 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. Inspired by the central limit theorem for G/G/m queuing servers under heavy load [23, 26], we were able to demonstrate [30] that in a load region of 80% 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 20% and 15% errors at load levels of 80% and 90%, respectively, for the cases (a) and (b) in the design space mentioned above. As our sensitivity analysis in Section 5 shows, such prediction errors can be well compensated for with no more than 5% and 3% resource overprovisioning at these load levels, respectively.

The work in this paper makes the following contributions. First, it generalizes the solution in [30] 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 solution to any white-box FJQNs at high load and hence, it makes a contribution to the queuing network theory as well. In fact, as we shall show in Section 4.1, for any white-box FJQN with M/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 solution is 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 ForkTail can lead to accurate resource provisioning for user-facing interactive datacenter services in a consolidated datacenter environment at high load. Finally, preliminary ideas are provided as to how to use this solution to facilitate tail-latency-SLO-guaranteed job scheduling and resource provisioning.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 introduces ForkTail and the empirical tail latency approximations. Section 4 performs extensive testing of the accuracy of the approximations. Section 5 presents the sensitivity analysis for ForkTail. Section 6 discusses how ForkTail may be used to facilitate effective job scheduling and resource provisioning with tail-latency-SLO guarantee. Finally, Section 7 concludes the paper and discusses future work.

2 RELATED WORK

Fork-Join structures are traditionally modeled by FJQNs. To date, the exact solution exists for a two-Fork-node FJQN only [18, 29]. Most works primarily focus on the approximation of mean response time [4, 29, 38] and its bounds [5, 12]. 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 [11, 29].

Some analytic results are available on redundant task issues [19, 32, 41]. They either address only a single replicated server subsystem with exponential task service time distribution [19] or parallel request load balancing without task spawning [32, 41].

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 [6] or M/M/k [25]. Computable stochastic bounds on request waiting and response time distributions for some FJQNs are provided in a recent work [35]. The most interesting and relevant work is given in [33]. The

authors of this work proposed a method 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. In Section 4.1, we apply our approach to these FJQNs, which are then compared against the approximations from this work, in terms of both prediction accuracies and computational complexity. 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.

Due to the lack of theoretical underpinning, the existing tail-latency-SLO-aware resource provisioning proposals cannot provide tail-latency SLO guarantee by design. Instead, various techniques such as tail-cutting techniques [14, 39], a combination of job priority and rate limiting based on network calculus [44] 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 [17, 40]. 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 200ms, i.e., probabilistically, only one out of 1000 requests should experience a response time greater than 200ms. 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 200ms. 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 200ms may be well within 100.

In summary, a solution that can predict the tail 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.

3 FORKTAIL

The black-box model described in this section, called ForkTail, greatly extends the scope of the black-box model introduced in [30] to address the entire design space mentioned in Section 1.

Consider a black-box Fork-Join model with each request or job (hereafter, these two terms are used interchangeably) in the incoming request flow spawning k tasks mapped to k out of N Fork nodes where $k \leq N$, 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). Requests arrive following a random arrival process with average arrival rate λ. Although the proposed solution applies to arbitrary arrival processes, we consider only Poisson arrival process, as it is

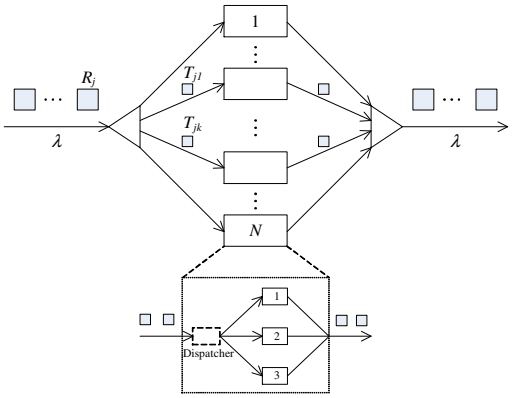

Figure 1: Black-box Fork-Join model.

widely recognized to be a reasonably accurate model for datacenter applications in practice [28]. 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 ForkTail 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 [36] and the references therein). In particular, a classic result, known as the central limit theorem for heavy traffic queuing systems [23, 26] states that for a G/G/m queue (i.e., general arrival process, general service time distribution, and m servers) 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. The intuition behind this approximation is that in the high load region, the long queuing effect helps effectively smooth out service time fluctuations (i.e., the law of large numbers), which causes the response time to converge to a distribution closely surrounding its mean value, i.e., the short-tailed exponential distribution, regardless of the actual arrival process and service time distribution. 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 can be approximated as a generalized exponential distribution function [20], 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 [20]

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

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

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. 2. 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 [30], we showed that this distribution significantly outperforms the exponential distribution in terms of tail latency predictive accuracy.

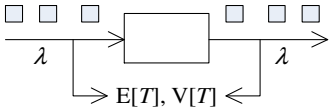

Figure 2: A Fork node as a black box.

Now, with all the Fork nodes in Fig. 1 being viewed as black boxes, the response time distribution for any request with k tasks can be approximated using the order statistics [37] 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. Our postulation is that as load reaches 80% or higher where precise resource provisioning is desirable and necessary, the tail-latency prediction errors introduced by this 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.

Tail latency x_p, defined as the pth percentile request response time, can then 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 request samples from the same service or different services with similar task service time statistics, thanks to its dependence on the first two moments of task response times only, i.e., the means and variances. Using the same example given in Section 2, 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 tail latency prediction, can be updated every tens of milliseconds, making it possible to use ForkTail to enable fast online tail-latency-guaranteed job scheduling and resource provisioning. This is in stark contrast to the 33-minute window required for the tail latency prediction based on direct tail-latency measurement.

The results so far is general, applying to the inhomogeneous 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 request to another or even for the two identical requests, 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 request 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 request response time for requests 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 request spawns k_i tasks.

Further assume that there are m request 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 requests 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).

3.1 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. As an example, we apply our approach to a large class of FJQNs, where each Fork node is an M/G/1 queue.

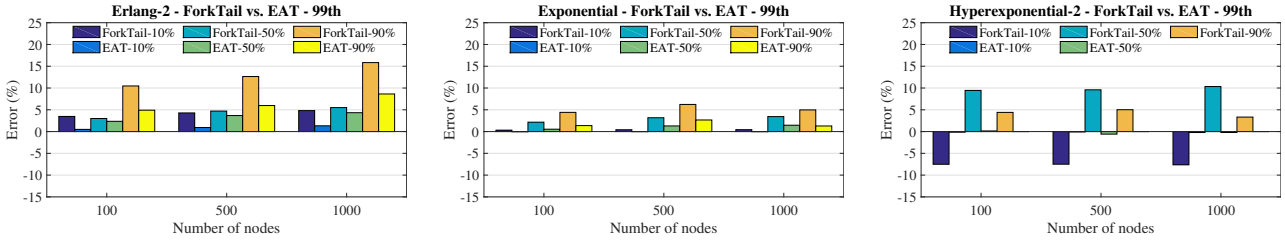

Figure 3: Prediction errors for the 99th percentile response times for ForkTail and EAT.

Let W, S, and T denote random variables for task waiting time, service time, and response time, respectively. Since W and S are independent random variables, the mean and variance of the task response time are given as,

$$\begin{aligned}\mathbb{E}[T] &= \mathbb{E}[W] + \mathbb{E}[S],\\ \mathbb{V}[T] &= \mathbb{V}[W] + \mathbb{V}[S].\end{aligned}$$

From Takács recurrence theorem [24], the kth moment of the waiting time can be computed by

$$\mathbb{E}[W^k] = \frac{\lambda}{1-\rho}\sum_{i=1}^{k}\binom{k}{i}\frac{\mathbb{E}[S^{i+1}]}{i+1}\mathbb{E}[W^{k-i}].$$

Using this theorem, the mean and variance of the task response time can be derived as follows,

$$\mathbb{E}[T] = \mathbb{E}[S]\left(1+\frac{\rho}{1-\rho}\cdot\frac{1+C_S^2}{2}\right), \tag{10}$$

$$\mathbb{V}[T] = \mathbb{E}[W]^2 + \frac{\lambda\mathbb{E}[S^3]}{3(1-\rho)} + \mathbb{E}[S^2] - \mathbb{E}[S]^2, \tag{11}$$

where $\mathbb{E}[S^k]$ is the kth moment of the service time; ρ is the server utilization, $\rho = \lambda\mathbb{E}[S]$; C_S^2 is the squared coefficient of variation of service times, $C_S^2 = \mathbb{V}[S]/\mathbb{E}[S]^2$; and $\mathbb{E}[W]$ is the mean waiting time, $\mathbb{E}[W] = \lambda\mathbb{E}[S^2]/[2(1-\rho)]$.

The task response time distribution can then be approximated by Eq. (1) whose parameters can be found by substituting Eqs. (10) and (11) into Eqs. (2) and (3), respectively. Finally, the tail latency for the homogeneous system can be obtained from Eq. (9).

4 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 $k = N$, $k \le N$, 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.,

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

Due to the page limitation, in this paper, we only provide testing results for the 99th percentile tail latencies. The testing results at the 99.9th percentile tail latencies are given in an extended version of this paper, which is available online [3]. All the conclusions drawn in this paper stay intact in [3].

4.1 Case 1: $k = N$

A notable example for this case is Web search engine [7] 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 Testing: Here we study the accuracy of ForkTail when applied to homogeneous, single-server-Fork-node Fork-Join systems with the assumption that the service time distribution is known in advance, the approach taken in previous works on performance analysis of FJQNs [37]. The tail latency prediction involves the following steps:

- Compute the mean, variance, and third moment for the given task service time distribution.
- Find the mean and variance of task response times from Eqs. (10) and (11).
- 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 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 prediction approach for *homogeneous* FJQNs [33], referred to as *efficient approximation for tails* (EAT). This approximation is based on analytical results from single-node and two-node systems. Fig. 3 shows the comparative results for three service time distributions studied in [33], i.e., Erlang-2, Exponential, and Hyperexponential-2, at the loads of 10%, 50%, and 90%[2] and numbers of nodes of 100, 500, and 1000.

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%, 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,

[2]For EAT, the case for Hyperexponential-2 at the load of 90% is not available, due to a numerical error running the code provided in [33].

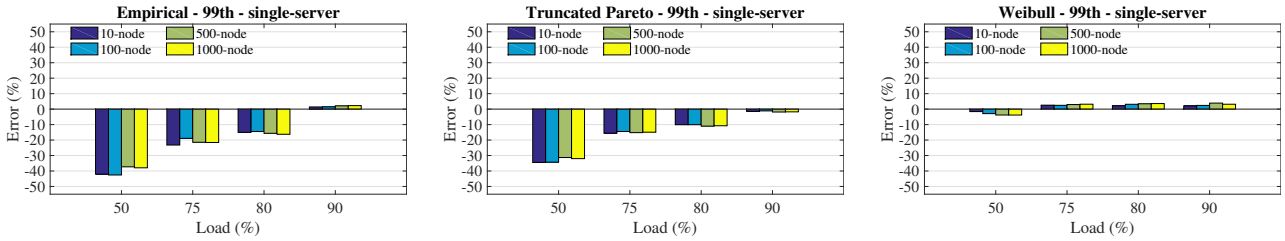

Figure 4: Prediction errors of the 99th percentile response times for white-box systems with single-server Fork nodes.

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 [21]. 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 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 [27] and cannot be easily dealt with by EAT, including the following,

- Empirical distribution measured from a Google search test leaf node provided in [27], which has a mean service time of 4.22ms, a coefficient of variance (CV) of 1.12, and the largest tail value of 276.6ms;
- Truncated Pareto distribution [21] with the same mean service time and a CV of 1.2, whose CDF is given by,

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

  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 [9], also with the same mean service time and a CV of 1.5, whose CDF is defined as,

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

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

Fig. 4 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%, for the entire load range studied, similar to the short-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% or higher, which is well within 17% and 5% at the load of 80% and 90%, respectively, agreeing with our postulation.

Black-Box Model-based Testing: 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 [14, 39]. 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 10ms, which is around the 95th percentile of the empirical distribution above.

Figs. 5–7 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. 5 are very close to those in Fig. 4. 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. 6 are also very close to those of the cases in Fig. 5, with errors being well within 20% and 10% at the loads of 80% and 90%, respectively, for all the case studies, further affirming our postulation. The two scenarios have similar performances 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. 7 shows that with the application of the tail-cutting technique, the prediction errors are substantially reduced, with less than 10% at the load of 80% or higher. This is consistent with the earlier observation, i.e., the shorter the tail, the smaller the prediction errors. This suggests that the tail-cutting techniques, often utilized in datacenters to curb the tail effects, can

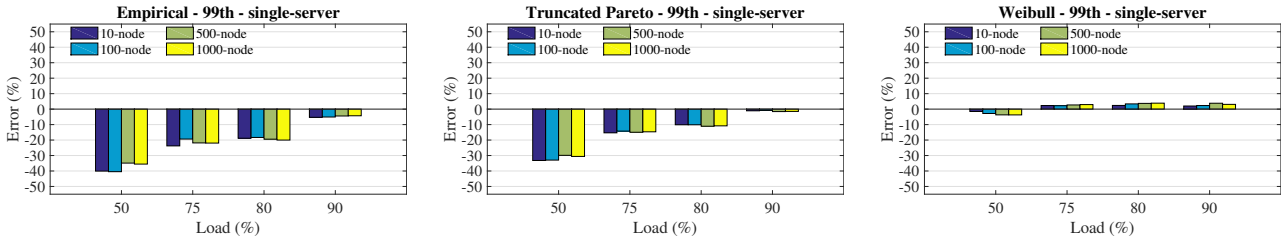

Figure 5: Prediction errors of the 99th percentile response times for black-box systems with single-server Fork nodes.

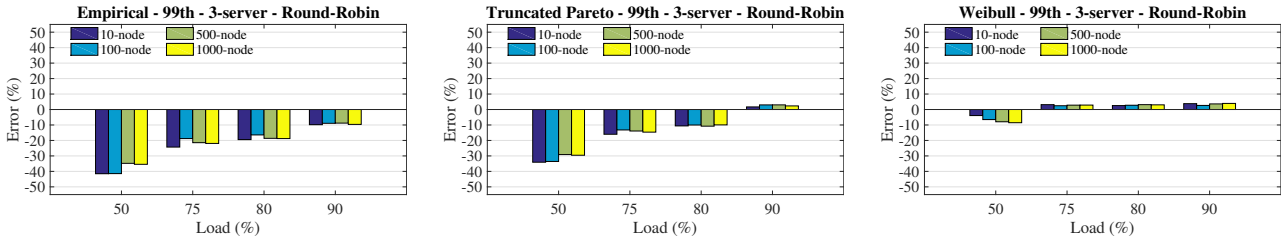

Figure 6: Prediction errors of the 99th percentile response times for black-box systems with 3-server Fork nodes and Round-Robin dispatching policy.

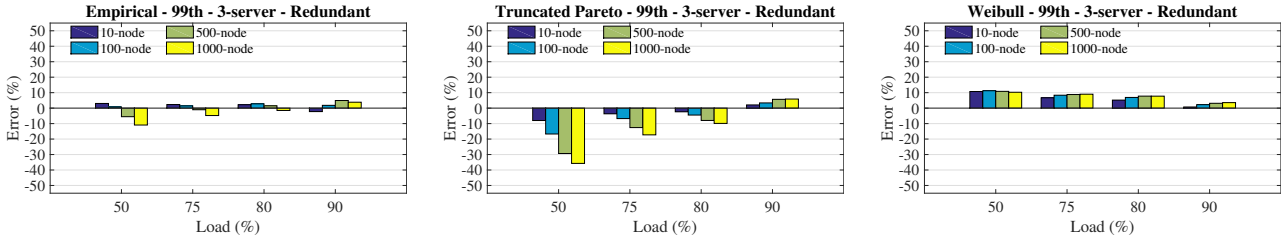

Figure 7: Prediction errors of the 99th percentile response times for black-box systems with 3-server Fork nodes and redundant-task-issue dispatching policy.

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) [2]. It looks up a keyword in a set of documents and returns the total number of lines containing that keyword, as depicted in Fig. 8. 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 128MB, corresponding to the default block size on HDFS (Hadoop Distributed File System) [1]. 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.

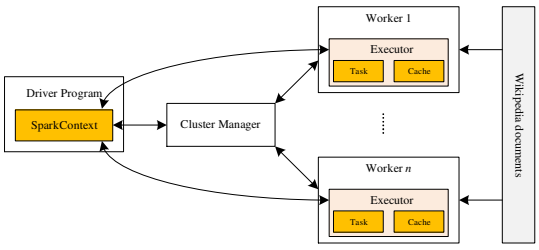

Figure 8: Experiment setup in Amazon EC2 cloud.

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

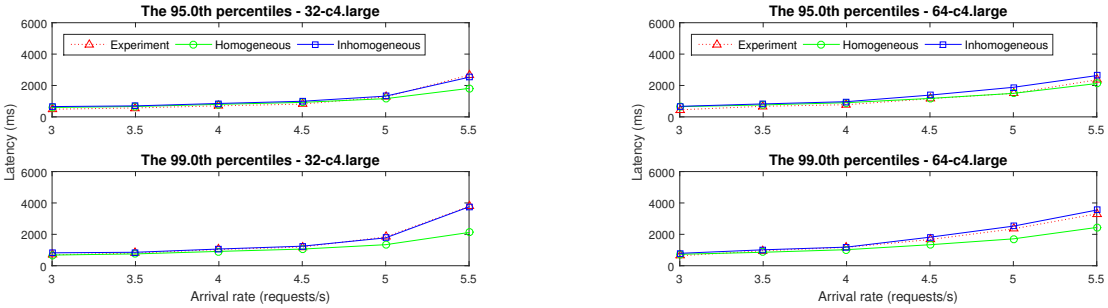

Figure 9: Predicted tail latencies for keyword occurrence counts in Amazon cloud with 32 (left) and 64 (right) nodes.

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 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 inhomogeneous model given in Eq. (4) is found to be more appropriate in high load regions. This observation is confirmed by the experimental results, presented in Fig. 9. As one can see, the inhomogeneous 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 inhomogeneous prediction, the prediction errors at both $N = 32$ and 64 and the 99th percentile are well within 10% in a high load region, i.e., 60% or higher. Note that the load here is measured in terms of request arrival rate. Since the system is inhomogeneous, 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. 1.

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.

Table 1: Estimated loads (%) for the testing cluster based on request arrival rates.

	Request arrival rates (requests/s)					
#workers	**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

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.

4.2 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 requests may spawn different numbers of tasks with distribution $P(K = k_i)$. Specifically, we study two scenarios where $P(K = k_i)$ is nonzero for a specific value of K and uniformly distributed. We further consider three different service time distributions: two heavy-tailed ones, the empirical and truncated Pareto as in Section. 4.1, and a light-tailed exponential distribution, with the same mean service time, i.e., 4.22ms.

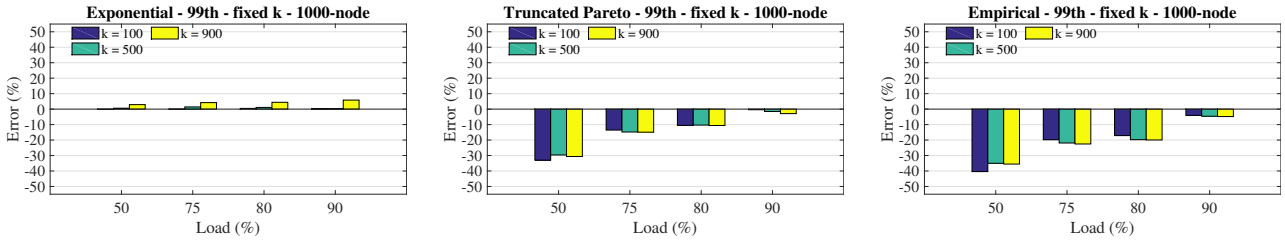

Figure 10: Prediction errors of the 99th percentile response times for an 1000-node cluster when the number of tasks per job is fixed ($k = 100, 500, 900$).

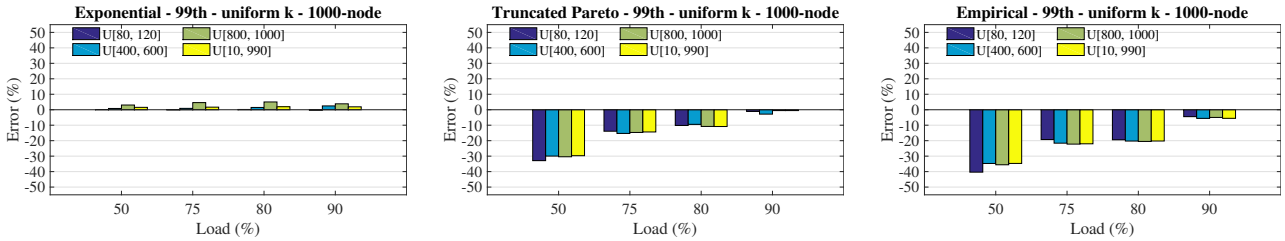

Figure 11: Prediction errors of the 99th percentile response times for an 1000-node cluster when the number of tasks per job is uniformly distributed.

Scenario 1: Fixed Number of Tasks per Request: In this scenario, we consider the cases when the number of forked tasks per request is a fixed number k ($k \leq N$), i.e., every incoming request is split into exactly k tasks which are dispatched to k randomly selected Fork nodes in an N-node cluster.

From Eqs. (6) and (8), we have,

$$F_X(x) = (1 - e^{-x/\beta})^{k\alpha}, \tag{12}$$

and the pth percentile is given by,

$$x_p = -\beta \log\left(1 - \left(\frac{p}{100}\right)^{1/k\alpha}\right). \tag{13}$$

Fig. 10 shows prediction errors for the 99th percentile response times for an 1000-node cluster with $k = 100$, 500, and 900 tasks. ForkTail provides good prediction in high load regions, with all the errors within 10% at the load of 90% and 20% at the load of 80% for all the cases studied. The case with the light-tailed exponential distribution gives quite accurate prediction for the entire range under study, all within 6%.

Scenario 2: Uniform Distribution: Here we deal with cases when an incoming request 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 $(a + b)/2$.

From Eqs. (6) and (8), we have,

$$F_X(x) = P \cdot \sum_{i=1}^{m} (1 - e^{-x/\beta})^{k_i \alpha}. \tag{14}$$

Fig. 11 presents prediction errors for an 1000-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 request response times when the system is under heavy load, i.e., 80% 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. 2 provides the predicted tail latencies for some given requests with distinct k values in a cluster of size 1000 and at the load of 90%. As one can see, the 99th percentile tail latencies for requests 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 request groups with each having k's in a small range. Table. 3 shows the accuracy of the prediction model at given k's, all well within 10% at load of 90%.

Table 2: The predicted 99th percentile of latencies (ms).

	Number of forked tasks				
Distribution	**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

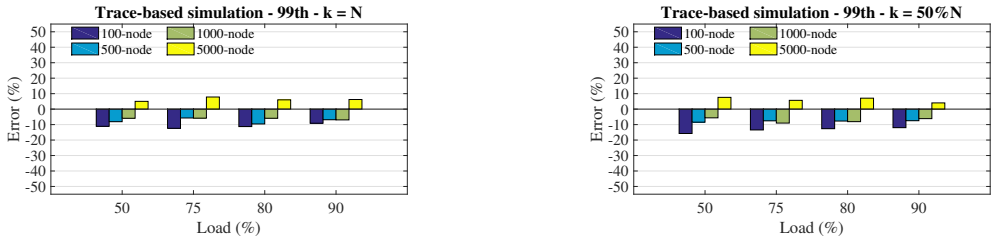

Figure 12: Prediction errors of the 99th percentile target response times in a consolidated workload environment when the tasks of each target job reach all the nodes (left plot) and randomly reach 50% number of nodes (right plot) in the cluster.

Table 3: Errors in the 99th percentile prediction when tracking jobs with a given number of tasks at load of 90%.

	Number of nodes				
Distribution	**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

4.3 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. 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 [13, 15, 43]. 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 [13, 15, 43]. 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 [43]. 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 [13]. Individual task times are drawn from a Normal distribution with the generated mean and a standard deviation that doubles the mean as in [15]. 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 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%, 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% 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, 1000, and 5000 nodes with each having three 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% are shown in Fig. 12. As one can see, the prediction errors are within 15% for all the cases studied.

5 SENSITIVITY ANALYSIS

From all the experiments above, we can see that the proposed model can be applied to a wide range of systems with reasonable prediction errors for the 99th percentiles, within 20% and 15% at the loads of 80% and 90%, 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 latency as a function of load.

We perform experiments with different load levels in the high load region, i.e., 78% to 95%, for FJQNs with different service time distributions, i.e., exponential, Weibull, truncated Pareto, and empirical. Fig. 13 shows results from both simulation and ForkTail for the 99th percentile response times for 1000-node systems. First, we note that ForkTail consistently overestimates the tail latency for the exponential and Weibull cases, while mostly underestimates it 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 tail latency at 90% load is roughly equal to the simulated one at 90.5% load. This means that ForkTail may lead to 0.5% resource over provisioning for the exponential cases. Following the same logic, it is easy to find that for both exponential and Weibull cases, ForkTail may result in no more than 1% resource overprovisioning in the entire 78%-95% load range. By the same token, we can find that for the truncated Pareto and empirical cases, ForkTail may cause up to 4% resource underprovisioning at 80% load and 2% at 90% load. This can be well compensated for by leaving a 4% resource margin in practice. This implies that in the worst-case when the actual service time

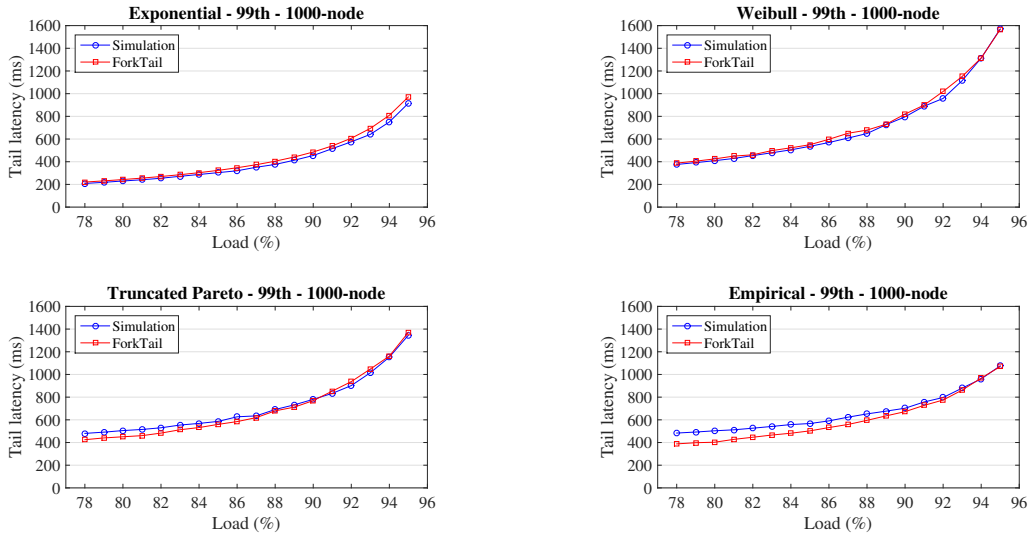

Figure 13: Differences in the 99th percentile response times from simulation and ForkTail for 1000-node systems with different service time distributions.

distribution is short-tailed, ForkTail may cause up to 5% and 3% resource overprovisioning at 80% and 90% load levels, respectively. This is tolerable, given that using our prediction model, we can improve datacenter resource utilization from currently under 50% all the way to 90% or higher.

Our sensitivity analyses for other Fork-Join structures, which are not shown here, have led to the similar conclusions. This means that ForkTail may serve as a powerful means to facilitate tail-latency-SLO-guaranteed job scheduling and resource provisioning for user-facing datacenter applications. The following section provides the preliminary ideas how this may be done.

6 FACILITATING RESOURCE PROVISIONING

In this section, we discuss how ForkTail may be used to facilitate both tail-latency-SLO-guaranteed job scheduling and resource provisioning. The proposed ideas are preliminary and somewhat sketchy, but yet, they do help reveal the promising prospects of ForkTail 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. 14. 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.

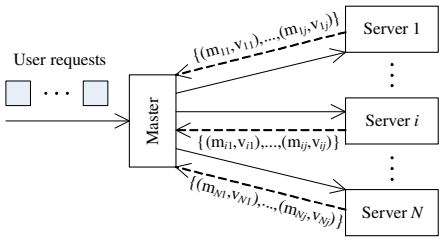

Figure 14: A hybrid, centralized-and-distributed job scheduler.

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, m, 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 serve 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 CONCLUSIONS AND FUTURE WORK

A key challenge in enabling tail-latency SLOs for user-facing data-center services and applications is how to predict the tail latency 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 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 an approximate tail latency solution for the black-box model. This approximation was found to be able to predict the tail latency for most practical scenarios consistently within 20% in a load region of 80% or higher, resulting in at most 5% 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, we shall develop both job scheduling and online/offline resource provisioning solutions with tail-latency SLO guarantee.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable feedback and helpful suggestions. This work is supported by the NSF under awards CCF XPS 1629625 and CCF 1704504.

REFERENCES

[1] Apache Hadoop. https://hadoop.apache.org.
[2] Apache Spark. https://spark.apache.org.
[3] Extended version. http://crystal.uta.edu/~hche/publications/forktail-ext.pdf.
[4] F. Alomari and D. A. Menasce. 2014. Efficient Response Time Approximations for Multiclass Fork and Join Queues in Open and Closed Queuing Networks. *IEEE Transactions on Parallel and Distributed Systems* 25, 6 (2014), 1437–1446.
[5] S. Balsamo, L. Donatiello, and N. M. Van Dijk. 1998. Bound performance models of heterogeneous parallel processing systems. *IEEE Transactions on Parallel and Distributed Systems* 9, 10 (1998), 1041–1056.
[6] S. Balsamo and I. Mura. 1995. Approximate response time distribution in Fork and Join systems. In *SIGMETRICS '95/PERFORMANCE '95*. 305–306.
[7] L. A. Barroso, J. Dean, and U. Hölzle. 2003. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro* 23, 2 (2003), 22–28.
[8] G. Blake and A. G. Saidi. 2015. Where does the time go? Characterizing tail latency in memcached. In *ISPASS '15*. 21–31.
[9] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. 2006. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. Wiley-Interscience.
[10] J. Brutlag. 2009. Speed Matters for Google Web Search. https://services.google.com/fh/files/blogs/google_delayexp.pdf.
[11] R. J. Chen. 2001. A hybrid solution of fork/join synchronization in parallel queues. *IEEE Transactions on Parallel and Distributed Systems* 12, 8 (2001), 829–845.
[12] R. J. Chen. 2011. An Upper Bound Solution for Homogeneous Fork/Join Queuing Systems. *IEEE Transactions on Parallel and Distributed Systems* 22, 5 (2011), 874–878.
[13] Y. Chen, S. Alspaugh, and R. Katz. 2012. Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads. *Proceedings of the VLDB Endowment* 5, 12 (Aug 2012), 1802–1813.
[14] J. Dean and L. A. Barroso. 2013. The Tail at Scale. *Commun. ACM* 56, 2 (2013), 74–80.
[15] P. Delgado, F. Dinu, A. M. Kermarrec, and W. Zwaenepoel. 2015. Hawk: Hybrid Datacenter Scheduling. In *USENIX ATC '15*. 499–510.
[16] C. Delimitrou and C. Kozyrakis. 2014. Quasar: Resource-Efficient and QoS-aware Cluster Management. In *ASPLOS '14*. 127–144.
[17] A. D. Ferguson, P. Bodik, E. Boutin, and R. Fonseca. 2012. Jockey: Guaranteed Job Latency in Data Parallel Clusters. In *EuroSys '12*. 99–112.
[18] L. Flatto and S. Hahn. 1984. Two Parallel Queues Created by Arrivals with Two Demands I. *SIAM J. Appl. Math.* 44, 5 (1984), 1041–1053.
[19] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytiä, and A. Scheller-Wolf. 2015. Reducing Latency via Redundant Requests: Exact Analysis. In *SIGMETRICS '15*. 347–360.
[20] R. D. Gupta and D. Kundu. 1999. Generalized Exponential Distributions. *Australian & New Zealand Journal of Statistics* 41, 2 (1999), 173–188.
[21] M. Harchol-Balter. 2013. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action* (1st ed.). Cambridge University Press.
[22] M. Jeon, S. Kim, S. Hwang, Y. He, S. Elnikety, A. L. Cox, and S. Rixner. 2014. Predictive Parallelization: Taming Tail Latencies in Web Search. In *SIGIR '14*. 253–262.
[23] J. F. C. Kingman and M. F. Atiyah. 1961. The single server queue in heavy traffic. *Proceedings of the Cambridge Philosophical Society* 57 (1961), 902–904.
[24] L. Kleinrock. 1975. *Queueing Systems, Vol. 1: Theory*. Wiley-Interscience.
[25] S. S. Ko and R. F. Serfozo. 2004. Response Times in M/M/s Fork-Join Networks. *Advances in Applied Probability* 36, 3 (2004), 854–871.
[26] J. Köllerström. 1974. Heavy Traffic Theory for Queues with Several Servers. I. *Journal of Applied Probability* 11, 3 (1974), 544–552.
[27] D. Meisner, W. Junjie, and T. F. Wenisch. 2012. BigHouse: A Simulation Infrastructure for Data Center Systems. In *ISPASS '12*. 35–45.
[28] D. Meisner, C. M. Sadler, A. L. Barroso, W. D. Weber, and T. F. Wenisch. 2011. Power Management of Online Data-Intensive Services. In *ISCA '11*. 319–330.
[29] R. Nelson and A. N. Tantawi. 1988. Approximate Analysis of Fork/Join Synchronization in Parallel Queues. *IEEE Trans. Comput.* 37, 6 (1988), 739–743.
[30] M. Nguyen, Z. Li, F. Duan, H. Che, Y. Lei, and H. Jiang. 2016. The Tail at Scale: How to Predict It?. In *USENIX HotCloud'16*.
[31] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani. 2013. Scaling Memcache at Facebook. In *NSDI '13*. 385–398.
[32] Z. Qiu and J. F. Perez. 2015. Evaluating the Effectiveness of Replication for Tail-Tolerance. In *CCGrid '15*. 443–452.
[33] Z. Qiu, J. F. Pérez, and P. G. Harrison. 2015. Beyond the Mean in Fork-Join Queues: Efficient Approximation for Response-Time Tails. *Performance Evaluation* 91 (2015), 99–116.
[34] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. 2012. Heterogeneity and Dynamicity of Clouds at Scale. In *SoCC '12*. 7:1–7:13.
[35] A. Rizk, F. Poloczek, and F. Ciucu. 2015. Computable Bounds in Fork-Join Queueing Systems. In *SIGMETRICS '15*. 335–346.
[36] S. Sani and O. A. Daman. 2014. Mathematical Modeling in Heavy Traffic Queuing Systems. *American Journal of Operations Research* 4 (2014), 340–350.
[37] A. Thomasian. 2014. Analysis of Fork/Join and Related Queueing Systems. *Comput. Surveys* 47, 2 (2014), 1–71.
[38] E. Varki. 2001. Response time analysis of parallel computer and storage systems. *IEEE Transactions on Parallel and Distributed Systems* 12, 11 (2001), 1146–1161.
[39] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. 2013. Low Latency via Redundancy. In *CoNEXT '13*. 283–294.
[40] A. Wang, S. Venkataraman, S. Alspaugh, R. Katz, and I. Stoica. 2012. Cake: Enabling High-level SLOs on Shared Storage Systems. In *SoCC '12*. 14:1–14:14.
[41] D. Wang, G. Joshi, and G. Wornell. 2014. Efficient Task Replication for Fast Response Times in Parallel Computation. *arXiv:1404.1328* (2014), 1–20.
[42] N. J. Yadwadkar, G. Ananthanarayanan, and R. Katz. 2014. Wrangler: Predictable and Faster Jobs using Fewer Resources. In *SoCC '14*. 26:1–26:14.
[43] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. 2010. Delay scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In *EuroSys '10*. 265–278.
[44] T. Zhu, A. Tumanov, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger. 2014. PriorityMeister: Tail Latency QoS for Shared Networked Storage. In *SoCC '14*. 29:1–29:14.