TAIL LATENCY PREDICTION FOR FORK-JOIN STRUCTURES

by

SAMI MARZOOK ALESAWI

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2020

TAIL LATENCY PREDICTION FOR FORK-JOIN STRUCTURES

The members of the Committee approve the doctoral dissertation of SAMI MARZOOK ALESAWI

Hao Che Supervising Professor	
Yonghe Liu	
Chengkai Li	
Bahram Khalili	
Dean of the Graduate School	

” People are of two kinds: either your brothers in Faith, or your EQUAL in Humanity.” - Imam Ali (A.S)

ACKNOWLEDGEMENTS

I want to take this chance to present my gratefulness and sincere gratitude to my supervising professor Dr. Hao Che who I'm honored to be one of his PhD students, for his continuous support and encouragement, and for his valuable feedbacks that I have received from him while doing my coursework and during my research projects as well. Also, I would like to thank the following professors: Dr. Yonghe Liu, Dr. Chengkai Li, and Dr. Bahram Khalili for their interest and valuable comments regarding my research and for accepting to assist as my dissertation committee.

Further, I would extend my gratitude to the Faculty of Computing and Information Technology in Rabigh, Department of Computer Science at King Abdulaziz University for granting me my scholarship and providing me with the financial support to pursue my doctoral studies. I am also grateful to all my teachers, instructors, coworkers who taught me and help improving my experiences during the years I had spent at schools whether in Saudi Arabia, or the United States.

Lastly, words never will be enough to express my absolute respect and love to my father and my brothers who have encouraged me, inspired and supported me all the way during my undergraduate and graduate studies. I am incredibly happy and grateful for the blessing of my mother, sisters, and my devoted wife for their sacrifice, cheering, and patience. Many thanks to my relatives and friends who helped by sharing me lots of those golden bits of advice from their own life experiences. I feel I am such a blessed and fortunate person who has all those chances worked together in motivating me to continue the unstoppable work and pursuing my future-dream until it became real.

July 9, 2020

ABSTRACT

TAIL LATENCY PREDICTION FOR FORK-JOIN STRUCTURES

SAMI MARZOOK ALESAWI, Ph.D.

The University of Texas at Arlington, 2020

Supervising Professor: Hao Che

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 dissertation, we propose and conduct a comprehensive study of a model for tail latency prediction for a wide range of fork-join structures of practical interests. The idea is to introduce a detailed examination of two main approaches that can be applied to our prediction model: 1) a black-box approach that covers a wide range of fork-join structures. And 2) a white-box approach for a wide range of fork-join queuing models. Our extensive testing results based on model-based and trace-driven simulations demonstrate that the model can consistently predict the tail latency within 20% and 15% prediction errors at 80% and 90% load levels, respectively. The experimental results confirmed the effectiveness of the prediction model in predicting tail latency at high load regions, making the model a valuable tool for resource provisioning and supporting scheduling decisions in datacenter clusters for guaranteeing users satisfaction.

Finally, the model is applied to the prediction of achievable resource utilization under any given tail-latency SLO. We derive a cluster load bound for a particular class of OLDI services. And for a simple Fork-Join queuing network model of M/M/1 Fork nodes, the derivation helps characterize the performance upper bound of such services, assuming that all the jobs have the same fanout degree. In general, due to the queuing effect, the tail latency is difficult to contain without throwing a significant amount of cluster resources to the problem, which would result in low cluster resource utilization.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 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 [2]. In a Fork-Join structure (e.g., Fig. 2.1), 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% [3].

Tail latency is considered to be the most important performance measure for user-facing datacenter applications [4] 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

[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.

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 [5, 6]. For example, 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 [5]. 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% [6]. 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 [6]. 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) [7]. 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 dissertation, 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 [8]. Although empirical solutions for some FJQNs are available, e.g., [9, 10, 11, 12, 13], 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) [7]; (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, 15] or fast recovery from straggling tasks [16]; (c) the case where the number of spawned tasks per request may vary from one request to another [17]; and (d) the case of consolidated services, where different types of services and applications may share the same datacenter cluster resources [18]. Clearly, the existing FJQNs can hardly cover such a design space in practice. Therefore, we provide comprehensive solutions to the above challenges, which will be explained within the next chapters. This dissertation makes the following major contributions:

(1) We proposed a black-box Fork-Join model that can be applicable to a wide range of Fork-Join structures of practical interests, making tail-latency prediction for OLDI applications a reality, chapter 2.

(2) With this model, the tail-latencies for a wide range of FJQNs are derived for the first time, making a contribution to the queuing theory as well, chapter 2.

(3) We demonstrate that this model can be used at consolidated environments to predict the tail of the target application among multiple existed applications, chapter 3.

(4) We present our analytical solution for tail prediction,i.e., the white-box approach, of both: single and consolidated applications, and have it available at the disposal to interested researchers, chapter 3.

(5) We demonstrate that this model can be extended to heterogeneous FJQNs. We apply variable fanout degrees, which is presented in chapter 2. And we push it further, in chapter 4, with applying different (or variant): load utilization, distributions, distributions-tails, or distributions service times, among nodes, where we find the tail-latency prediction error stays within less than 15%.

(6) As an application of our proposed solution:

– We present our method to translate any given users requirements (SLOs) into task performance budgets, which we do believe it will advance the current tasks scheduling algorithms into a further level, for bettering user experience, chapter 4.

– We present our derived method to calculate the load bound for a particular class of OLDI services, chapter 5.

Finally, the research works included in this dissertation are organized as the following:

1.1 Black-BOX Fork-Join Model

In chapter 2, we propose a solution for the *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. As we shall see in this study, the proposed black-box model indeed adequately covers the most design space, if not all. Comprehensive testing and verification of the proposed solution are performed for all Fork-Join structures. Also, in this chapter, sensitivity analysis is provided, which indicates that ForkTail can lead to accurate resource provisioning for user-facing interactive datacenter services in a consolidated datacenter environment at a high load. Preliminary ideas are introduced as to how to use this solution to facilitate tail-latency-SLO-guaranteed job scheduling and resource provisioning.

1.2 White-BOX Model for Consolidated Applications

Consolidating applications in datacenters becomes a necessity to reduce cost and improve the return on investment by increasing the utilization and allowing resource sharing among different applications[19]. This comes at a price of poor user experience and high delays in processing user requests. Unfortunately, many research works primarily considered only a flow of jobs from a single application. The work in chapter 2 included the approximation for the tail latency of consolidated applications but based on a black-box approach, which approximates the task response time distributions using the measured means and variances of task response times on the Fork nodes. To the best of our knowledge, no previous work attempts to provide an analytical solution for FJQNs with consolidated workloads. This is the primary motivation for the work in chapter 3. In chapter 3, we propose a closed-form solution, i.e., *a white-box approach*, to the approximation of the tail latency of a given target application in FJQNs with *a mixture of applications*, each following a different service time distribution.

1.3 Heterogeneous FJQNs

Nowadays, many applications' requests poured into common heterogeneous infrastructure shared among applications. Therefore, heterogeneity in resources is also known to be one of the leading causes of the high variability among task response times. Lack of having enough studies for tail prediction at inhomogeneous environments is the main reason that motivates the work in chapter 4. In addition, three new novel ideas contributed to the research in chapter 4: 1) convergence point for reliable metrics, 2) resource usage predictability, and 3) budget translation; which render the uniqueness of this work as it takes further steps toward providing a practical, reliable, and budget guaranteed scheduling mechanisms in the foreseeable future.

1.4 Achievable Cluster Utilization for OLDI applications

All the provided works in previous chapters were just a continuation of the model verification at every probable circumstance that might exist. However, in this chapter 5, we depart from that context and move toward covering one of its possible applications by providing an answer for one crucial question, which is: *Is Datacenter Resource Overprovisioned to Support OLDI Services?* We are going to introduce our derived model for estimating the achievable load level based on a given target-ratio between the desired tail latency SLO and the average service time. As a beginning, we provide our numerical analysis of the derived model. Then we verify the correctness of the derived model by comparing with several results of simulation experiments, as well as numerical comparison with the GE prediction model. Lastly, we provide intuition about the model's effectiveness at predicting the upper bound at different workloads conditions.

CHAPTER 2

Black-BOX Fork-Join Model

To tackle the challenges mentioned previously in the introduction, in this chapter, 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 [20, 21], we were able to demonstrate [22] 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 2.3 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 chapter 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 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 2.2.1, 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 chapter is organized as follows. Section 2.1 introduces ForkTail and the empirical tail latency approximations. Section 2.2 performs extensive testing of the accuracy of the approximations. Section 2.3 presents the sensitivity analysis for ForkTail. Section 2.4 discusses how ForkTail may be used to facilitate effective job scheduling and resource provisioning with tail-latency-SLO guarantee. Section 2.5 reviews the related work. Finally, Section 2.6 concludes the chapter and discusses future work.

2.1 ForkTail

The black-box model described in this section, called ForkTail, greatly extends the scope of the black-box model introduced in [22] to address the entire design space mentioned in Chapter 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. 2.1. The results from all k tasks

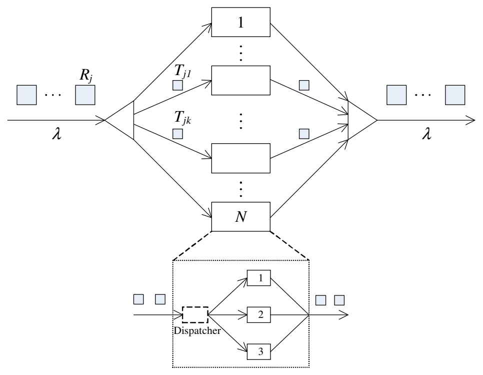


Figure 2.1: Black-box Fork-Join model.

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 widely recognized to be a reasonably accurate model for datacenter applications in practice [23]. 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. 2.1.

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 [24] 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 (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 [25], as follows,

$$F_T(x) = (1 - e^{-x/\beta})^{\alpha}, \quad x > 0, \alpha > 0, \beta > 0, \tag{2.1}$$

where α and β are shape and scale parameters, respectively.
The mean and variance of the task response time are given by [25]

$$\mathbb{E}[T] = \beta[\psi(\alpha + 1) - \psi(1)], \tag{2.2}$$

$$\mathbb{V}[T] = \beta^2[\psi'(1) - \psi'(\alpha + 1)], \tag{2.3}$$

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

From Eqs. (4.2) and (4.3), it is clear that the distribution in Eq. (4.1) 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. 2.2. 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 [22], we showed that this distribution significantly outperforms the exponential distribution in terms of tail latency predictive accuracy.

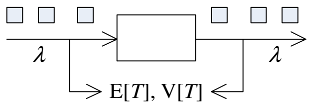


Figure 2.2: A Fork node as a black box.

Now, with all the Fork nodes in Fig. 2.1 being viewed as black boxes, the response time distribution for any request with k tasks can be approximated using the order statistics [8] as follows,

$$F_X^{(k)}(x) = \prod_{i=1}^{k} F_{T_i}(x) = \prod_{i=1}^{k} (1 - e^{-x/\beta_i})^{\alpha_i}, \tag{2.4}$$

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 chapter 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,

$$x_p = F_X^{(k)^{-1}}(p/100). \tag{2.5}$$

Eq. (4.5) 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 2.5, 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. (4.5) 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. (4.5) is a fine-grained tail latency expression. For certain applications, such as offline resource provisioning (see Section 2.4 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,

$$F_X^{(k)}(x) = (1 - e^{-x/\beta})^{k\alpha}. \tag{2.6}$$

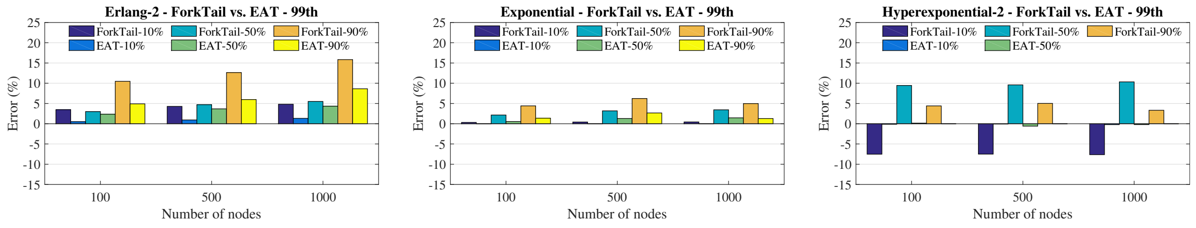


Figure 2.3: Prediction errors for the 99th percentile response times for ForkTail and EAT.

This is because the means and variances given in Eqs. (4.2) and (4.3) are the same for the homogeneous case. A coarser-grained cumulative distribution function (CDF) of the request response time can then be written as,

$$F_X(x) = \sum_{k_i} F_{X|K}(x|k_i)P(K = k_i), \tag{2.7}$$

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. (2.6), 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,

$$F_X(x) = \sum_{i=1}^{m} P_i \cdot F_X^{(k_i)}(x). \tag{2.8}$$

Correspondingly, the tail latency for the m groups of requests as a whole can then be readily obtained, similar to Eq. (4.5), as follows,

$$x_p = F_X^{-1}(p/100). \tag{2.9}$$

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. (2.9).

2.1.1 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.

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 [26], 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,

$$\mathbb{E}[T] = \mathbb{E}[S]\left(1 + \frac{\rho}{1-\rho}\cdot\frac{1+C_S^2}{2}\right), \tag{2.10}$$

$$\mathbb{V}[T] = \mathbb{E}[W]^2 + \frac{\lambda\mathbb{E}[S^3]}{3(1-\rho)} + \mathbb{E}[S^2] - \mathbb{E}[S]^2, \tag{2.11}$$

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. (4.1) whose parameters can be found by substituting Eqs. (2.10) and (2.11) into Eqs. (4.2) and (4.3), respectively. Finally, the tail latency for the homogeneous system can be obtained from Eq. (2.9).

2.2 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.,

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

In this chapter, 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 work, which is available online [27]. All the conclusions drawn in this chapter stay intact in [27].

2.2.1 Case 1: $k = N$

A notable example for this case is Web search engine [28] 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 [8]. The tail latency prediction involves the following steps:

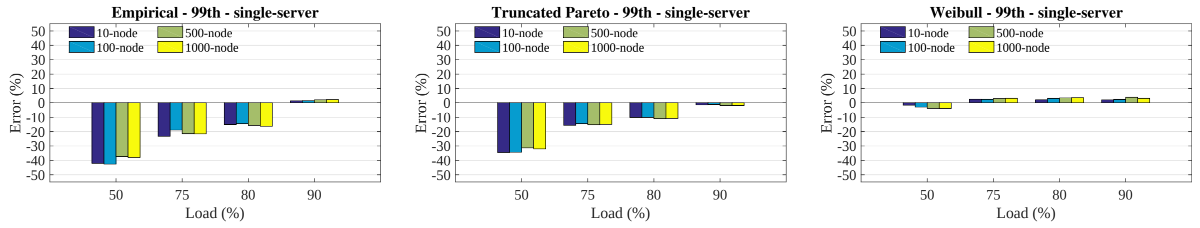


Figure 2.4: Prediction errors of the 99th percentile response times for white-box systems with single-server Fork nodes.

– 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. (2.10) and (2.11).

– Substitute the above mean and variance into Eqs. (4.2) and (4.3), respectively, and solve that system of equations to find the scale and shape parameters of the generalized exponential distribution in Eq. (4.1), which is used to approximate the task response time distribution.

– Calculate the pth percentile of request response times from Eq. (2.9).

First, we compare ForkTail against the state-of-the-art prediction approach for *homogeneous* FJQNs [13], referred to as *efficient approximation for tails* (EAT). This approximation is based on analytical results from single-node and two-node systems. Fig. 2.3 shows the comparative results for three service time distributions studied in [13], i.e., Erlang-2, Exponential, and Hyperexponential-2, at the loads of 10%, 50%, and 90%[1] 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

[1]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 [13].

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 [29]. 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 [30] and cannot be easily dealt with by EAT, including the following,

– Empirical distribution measured from a Google search test leaf node provided in [30], 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 [29] with the same mean service time and a CV of 1.2, whose CDF is given by,

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

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

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

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

Fig. 2.4 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. (4.2) and (4.3), respectively, to find the shape and scale parameters, which are in turn used to predict the tail latency based on Eq. (2.9).

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, 15]. 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

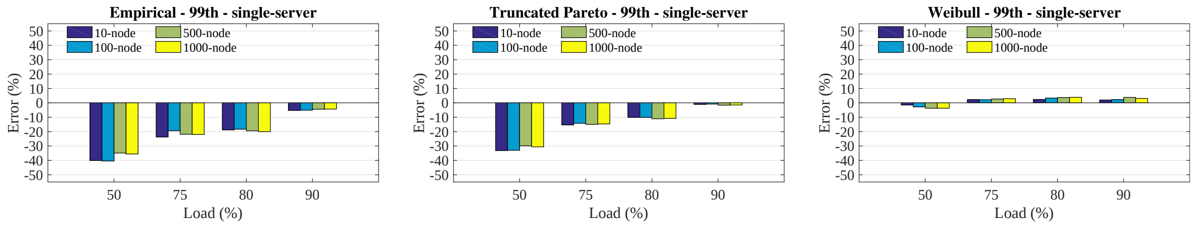


Figure 2.5: Prediction errors of the 99th percentile response times for black-box systems with single-server Fork nodes.

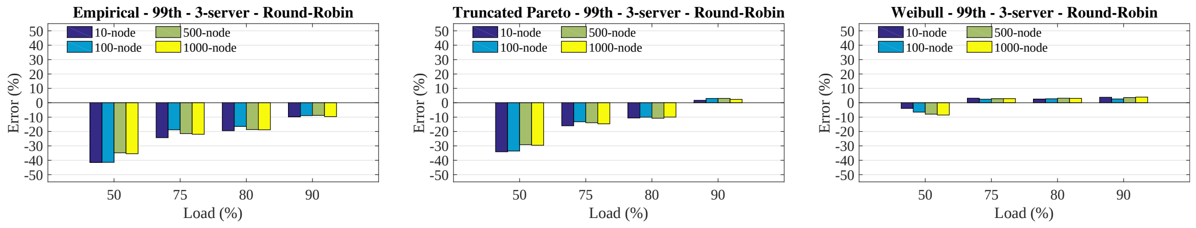


Figure 2.6: Prediction errors of the 99th percentile response times for black-box systems with 3-server Fork nodes and Round-Robin dispatching policy.

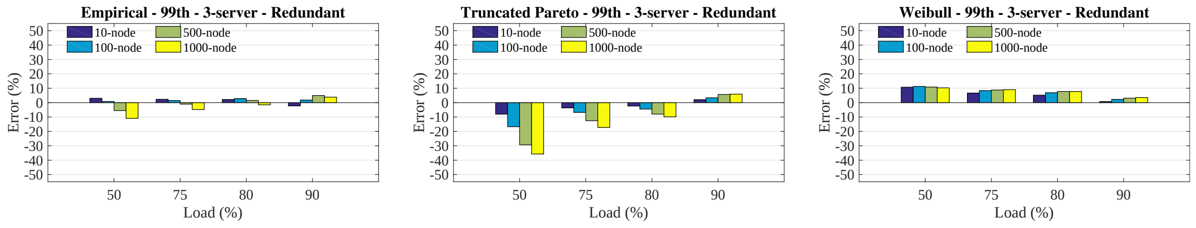


Figure 2.7: Prediction errors of the 99th percentile response times for black-box systems with 3-server Fork nodes and redundant-task-issue dispatching policy.

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[2].

Figs. 2.5–2.7 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

[2]As the $10ms$ in Truncated Pareto service time distribution is at the 40%tile, in Figure 2.7, the algorithm at larger cluster sizes is expected to increase the number of issued copies, which the reason for the increase in errors. Hence, it is a matter of how to apply the technique to suit each specific environment's needs based on its performance statistics.

node, respectively. First, we note that the prediction errors for the cases in Fig. 2.5 are very close to those in Fig. 2.4. 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. 2.6 are also very close to those of the cases in Fig. 2.5, 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. 2.7 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 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) [31]. It looks up a keyword in a set of documents and returns the total number of lines containing that keyword, as depicted in Fig. 2.8. 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) [32]. 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.

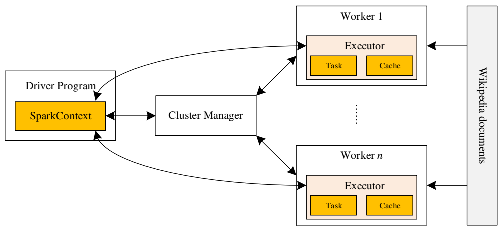


Figure 2.8: Experiment setup in Amazon EC2 cloud.

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 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. (4.1). 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

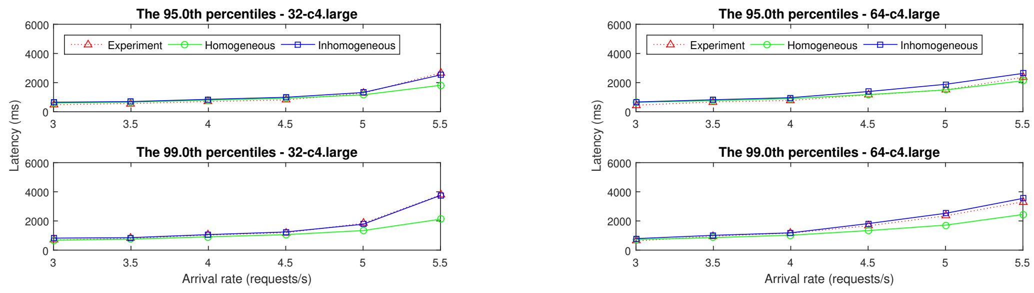


Figure 2.9: Predicted tail latencies for keyword occurrence counts in Amazon cloud with 32 (left) and 64 (right) nodes.

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. (2.4) is found to be more appropriate in high load regions. This observation is confirmed by the experimental results, presented in Fig. 2.9. 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. 2.1.

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,

Table 2.1: 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

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.

2.2.2 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. (2.8) and (2.9)) 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

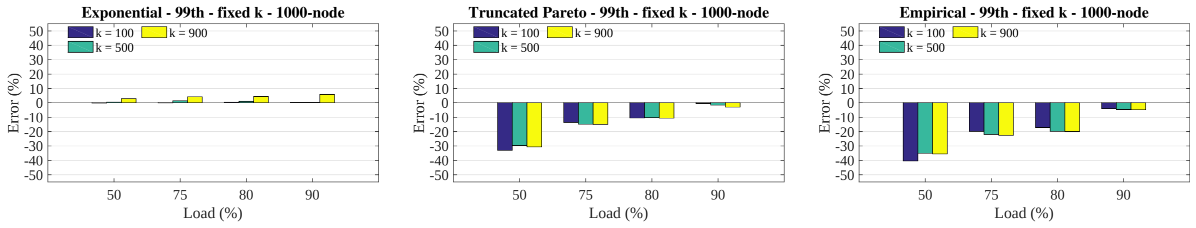


Figure 2.10: Prediction errors of the 99th percentile response times for an 1000-node cluster when the number of tasks per job is fixed ($k = 100, 500, 900$).

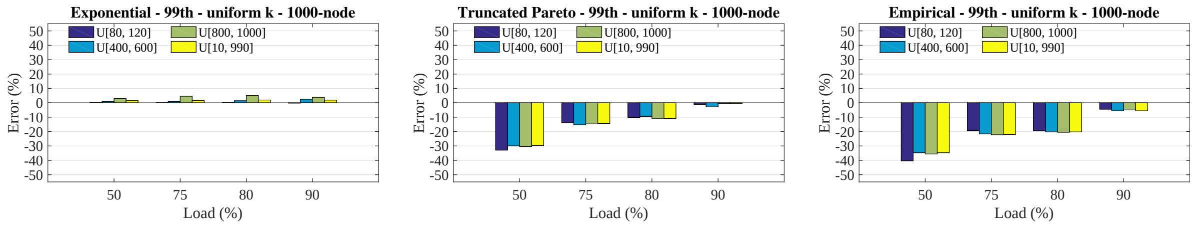


Figure 2.11: Prediction errors of the 99th percentile response times for an 1000-node cluster when the number of tasks per job is uniformly distributed.

distributions: two heavy-tailed ones, the empirical and truncated Pareto as in Section. 2.2.1, and a light-tailed exponential distribution, with the same mean service time, i.e., 4.22ms.

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.

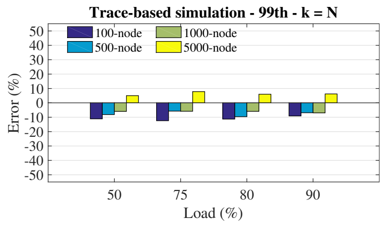
From Eqs. (2.6) and (2.8), we have,

$$F_X(x) = (1 - e^{-x/\beta})^{k\alpha}, \tag{2.12}$$

and the pth percentile is given by,

$$x_p = -\beta \log \left(1 - \left(\frac{p}{100} \right)^{1/k\alpha} \right). \tag{2.13}$$

Fig. 2.10 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



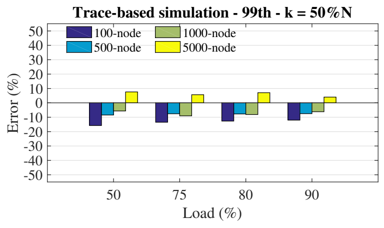


Figure 2.12: Prediction errors of the 99th percentile target response times in a consolidated workload environment when the tasks of each target job reach all the nodes (left plot) and randomly reach 50% number of nodes (right plot) in the cluster.

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. (2.6) and (2.8), we have,

$$F_X(x) = P \cdot \sum_{i=1}^{m} (1 - e^{-x/\beta})^{k_i \alpha}. \tag{2.14}$$

Fig. 2.11 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. 2.2 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. 2.3 shows the accuracy of the prediction model at given k's, all well within 10% at load of 90%.

Table 2.2: 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

Table 2.3: 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

2.2.3 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 [33, 34, 18]. 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 [33, 34, 18]. 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 [34]. 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 [33]. Individual task times are drawn from a Normal distribution with the generated mean and a standard deviation that doubles the mean as in [18]. 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

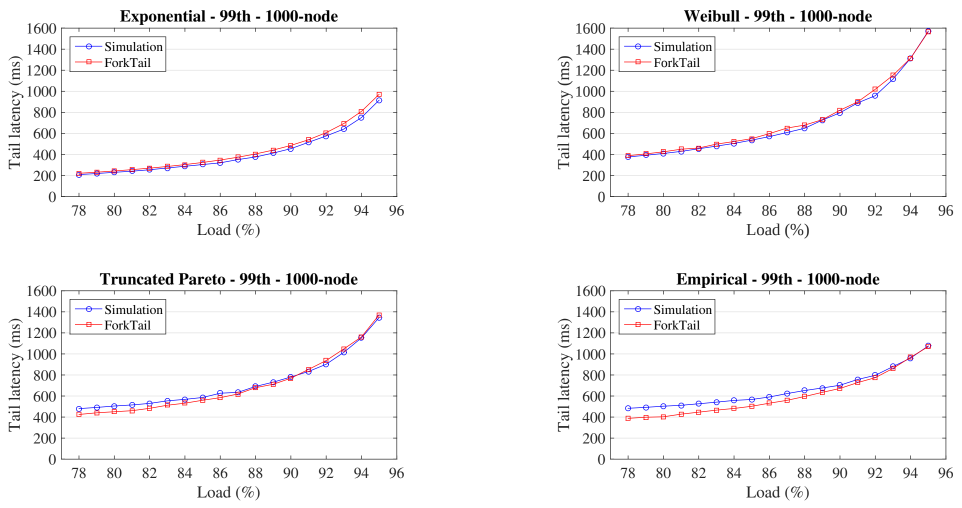


Figure 2.13: Differences in the 99th percentile response times from simulation and ForkTail for 1000-node systems with different service time distributions.

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. 2.12. As one can see, the prediction errors are within 15% for all the cases studied.

2.3 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. 2.13 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 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.

2.4 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. 2.14. 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. (4.5), 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.

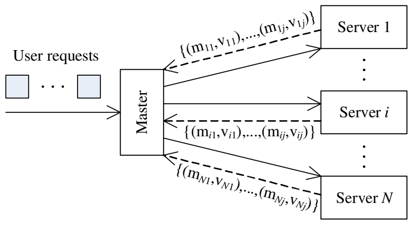

User requests
Master
Server 1
Server i
Server N
$\{(m_{11},v_{11}),...,(m_{1j},v_{1j})\}$
$\{(m_{i1},v_{i1}),...,(m_{ij},v_{ij})\}$
$\{(m_{N1},v_{N1}),...,(m_{Nj},v_{Nj})\}$

Figure 2.14: A hybrid, centralized-and-distributed job scheduler.

Resource Provisioning: ForkTail for the homogeneous case (i.e., Eqs. (2.8) and (2.9)) 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. (2.9) 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.

2.5 Related Work

Fork-Join structures are traditionally modeled by FJQNs. To date, the exact solution exists for a two-Fork-node FJQN only [35, 9]. Most works primarily focus on the approximation of mean response time [9, 10, 36] and its bounds [37, 38]. 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 [9, 12].

Some analytic results are available on redundant task issues [39, 40, 41]. They either address only a single replicated server subsystem with exponential task service time distribution [40] or parallel request load balancing without task spawning [39, 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 [42] or M/M/k [11]. Computable stochastic bounds on request waiting and response time distributions for some FJQNs are provided in a recent work [43]. The most interesting and relevant work is given in [13]. 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 2.2.1, 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, 15], 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 [45, 46]. 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.

2.6 Conclusions and Future Work

A key challenge in enabling tail-latency SLOs for user-facing datacenter 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 chapter, 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.

CHAPTER 3

White-BOX Model for Consolidated Applications

3.1 Using Analytic Solution For Consolidated Environment

Parallel computing has become predominant in datacenters nowadays due to ever-increasing amount of data to be processed in datacenter applications. Such applications usually need to be split into smaller tasks that are executed concurrently on hundreds or thousands machines for faster response time. While parallel computing can improve responsiveness and scalability, it makes effective job scheduling and resource provisioning extremely challenging. For example, the run-time variabilities of distributed task execution times, especially in the presence of synchronization barriers, can result in highly variable job completion times.

Similarly, consolidating applications in datacenters becomes a necessity to reduce cost and improve the return on investment by increasing the utilization and allowing resource sharing among different applications[19]. Unfortunately, this comes at a price of poor user experience and high delays in processing user requests. Therefore, maintaining a good user experience with guaranteed low latency in response time has become the current trend in datacenters [47, 48]. This newly arising demand for providing users with guaranteed service level objectives (SLOs) provoked lots of researchers to engineer many novel solutions to this challenging problem[49, 50, 51]. However, due to the lack of a performance model for the problem, a common practice today for datacenter applications is to overprovision resources to ensure that a job can finish within the allocated time slot or meet a predefined SLO in terms of, e.g., tail latency. Consequently, to meet the performance targets for applications, today's datacenters typically run at 10%–50% of their capacities [5, 6]. This may result in high costs from both user's and service provider's points of view.

Fork-Join structures are basic building blocks that underlay the workflows of many datacenter applications, e.g., web search, machine-translation, and social networking [47]. In these structures, an incoming job spawns multiple tasks which are processed by multiple processing nodes in the system. The job is considered to be completed when all of its tasks are finished and all the partial results are merged, which is called *barrier synchronization*. Therefore, the slowest task determines the response time of such a job. This processing pattern exactly follows the classical Fork-Join queuing network (FJQN) model, which is notoriously hard to solve in queuing theory [8]. Most of previous works on FJQNs in the literature attempt to find the approximation for job mean response time [9, 10, 36, 12] and its bounds [37, 38]. With the emergence of user-facing latency-sensitive applications, tail latency has drawn more attentions recently. Several recent works [13, 43, 52, 53] attempted to provide analytical solutions for the tail latency in FJQNs. However, they primarily considered only a flow of jobs from a single application. The work in [53] included the approximation for the tail latency of consolidated applications but based on a black-box approach, which approximates the task response time distributions using the measured means and variances of task response times on the Fork nodes.

To the best of our knowledge, no previous work attempts to provide an analytical solution for FJQNs with consolidated workloads. This is the primary motivation for this research work. In this chapter, we propose a closed-form solution, i.e., *a white-box approach*, to the approximation of the tail latency of a given target application in FJQNs with *a mixture of applications*, each following a different service time distribution. We will elaborate more on the effect of consolidated workloads on the accuracy of the prediction model through detailed scenarios, considering different distributions for different applications and different percentages of target application against the background applications. This was not adequately covered in [53] where only a trace-driven simulated result based on black-box measurements was presented as a shrunken section in the published work. For all the cases studied, the validation against simulation results shows that the proposed model yields all the prediction errors well within 10% at the load of 75% or higher. This indicates the effectiveness of the proposed model in predicting tail latency for a target application in a consolidated environment at high load regions, where resource provisioning is most desirable.

The rest of the chapter is organized as follows. Section 3.2 presents the proposed prediction model for the tail latency of a target application in a mixture of consolidated applications. Section 3.3 shows experimental results for different scenarios of the consolidated workloads. Section 3.4 reviews related work. Finally, Section 3.5 concludes the chapter and discusses future work.

3.2 The Proposed Model

In this section, we present our proposed analytical solution to the approximation of the tail latency of a target application in a consolidated environment, i.e., a mixture of applications sharing common datacenter resources. We shall derive a closed-form approximate solution for the pth percentile of a target (or tagged) application in the mixture.

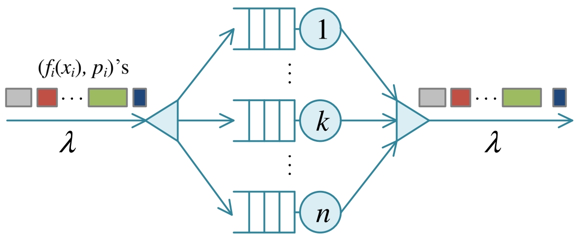


Figure 3.1: A Fork-Join model with each node as an M/G/1 queue.

Consider a system running a mixture of applications $A_1, A_2, \ldots, A_m$ with corresponding weights $p_1, p_2, \ldots, p_m$ in the mixture as in Fig. 3.1. The applications flow into the system with an average arrival rate of λ.

Assume that service times of application A_i follow distribution $f_i(x)$, i.e., $X_i \sim f_i(x)$, whose mean and variance are μ_i and σ_i^2, respectively.

Let X be a random variable representing service times on each node in the system. For a long runtime and load balanced between nodes, the effective service time distribution puts on each node can be viewed as a mixture of individual distributions with their corresponding weights, i.e.,

$$X \sim f(x) = \sum_{i=1}^{m} p_i \cdot f_i(x). \tag{3.1}$$

The kth moment of X can be written as

$$\begin{aligned} \mu^{(k)} = \mathbb{E}[X^k] &= \sum_{i=1}^{m} p_i \cdot \mathbb{E}[X_i^k] \\ &= \sum_{i=1}^{m} p_i \cdot \mu_i^{(k)} = \mathbb{E}[\mathbb{E}[X^k|\mu_i]], \end{aligned} \tag{3.2}$$

where $\mu_i^{(k)}$ is the kth moment of X_i.

According to the law of total variance, the variance of X is given by

$$\begin{aligned} Var(X) &= \mathbb{E}[Var(X|\mu_i)] + Var(\mathbb{E}[X|\mu_i]) \\ &= \sum_{i=1}^{m} p_i \cdot \sigma_i^2 + \sum_{i=1}^{m} p_i \cdot \mu_i^2 - \left(\sum_{i=1}^{m} p_i \cdot \mu_i\right)^2 \end{aligned} \tag{3.3}$$

In this work, we assume that the applications arrive at the system following Poisson process, which is a reasonably acceptable model for datacenter applications in practice [54], and their tasks are randomly distributed to the Fork nodes. Therefore, each Fork node can be viewed as an M/G/1 queue system, i.e., a Poisson arrival process with a general service time distribution and one service center.

The first and second moments of the task waiting time at each Fork node, i.e., an M/G/1 queuing system, are given as follows [26],

$$\mathbb{E}[W] = \frac{\lambda \mathbb{E}[X^2]}{2(1-\rho)} = \frac{\rho \mathbb{E}[X]}{1-\rho} \left(\frac{1 + C_X^2}{2} \right), \tag{3.4}$$

$$\mathbb{E}[W^2] = 2\mathbb{E}[W]^2 + \frac{\lambda \mathbb{E}[X^3]}{3(1-\rho)}, \tag{3.5}$$

where $\mathbb{E}[X^k]$ is the kth moment of the service time of the mixture given by Eq. (3.2); $C_X^2 = Var(X)/\mathbb{E}[X]^2$ is the squared coefficient of variation with $Var(X)$ being the variance given by Eq. (3.3); and $\rho = \lambda \mathbb{E}[X]$ is the utilization or load on each Fork node at average arrival rate λ.

Therefore, the mean and variance of response times of the target task at each Fork node can be written as,

$$\mathbb{E}[T] = \mathbb{E}[W] + \mathbb{E}[X_t], \tag{3.6}$$

$$\begin{aligned} Var(T) &= Var(W) + Var(X_t), \\ &= (\mathbb{E}[W^2] - \mathbb{E}[W]^2) + (\mathbb{E}[X_t^2] - \mathbb{E}[X_t]^2), \end{aligned} \tag{3.7}$$

where $\mathbb{E}[X_t]$ and $\mathbb{E}[X_t^2]$ are the first and second moments of the target task service time; and $\mathbb{E}[W]$ and $\mathbb{E}[W^2]$ are given in Eqs. (3.4) and (3.5).

It was proven that the waiting time distribution for G/G/m queue systems at heavy traffic conditions converges to an exponential distribution [55, 56]. Inspired by this result, the work in [52] postulated that the response time distribution can be approximated by a generalized exponential distribution as in Eq. (4.1), which outperforms the exponential distribution in term of tail latency prediction at high load regions for a wide range of service time distributions, including light-tailed, heavy-tailed, and empirical distributions,

$$F_T(x) = (1 - e^{-x/\beta})^{\alpha}, \quad x > 0, \alpha > 0, \beta > 0, \tag{3.8}$$

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

Similarly, in this work, the response time distribution of the target task on each Fork node is also approximated by the generalized exponential distribution as in Eq. (4.1), whose mean and variance are given by [57],

$$\mathbb{E}[T] = \beta[\psi(\alpha+1) - \psi(1)], \tag{3.9}$$

$$Var(T) = \beta^2[\psi'(1) - \psi'(\alpha+1)], \tag{3.10}$$

where $\psi(.)$ and its derivative are the digamma and polygamma functions. Given service time distribution $f_i(x_i)$[1] and weight p_i $(i = 1, 2, \dots, m)$ for each application, one can find parameters α and β for the target task by plugging the calculated mean and variance from Eqs. (3.6) and (3.7) into Eqs. (4.2) and (4.3), respectively, and solving this system of equations.

Assume that the jobs from the target application is forked to k nodes $(1 \leq k \leq N)$ with corresponding target task response time distributions $F_{T_j}(x)$'s $(j = 1, 2, \dots, k)$, as in [53], the system response time distribution for the target application can be approximated as,

$$\begin{aligned} F(x) &= P(\max_{1\leq j\leq k} T_j \leq x) \\ &\approx \prod_{j=1}^{k} F_{T_j}(x) = \prod_{j=1}^{k}(1 - e^{-x/\beta_j})^{\alpha_j}, \end{aligned} \tag{3.11}$$

and the pth percentile can be written as,

$$x_p = F^{-1}(p/100). \tag{3.12}$$

In case all the Fork nodes are homogeneous, the target response time distribution can be simplified as,

$$F(x) = (1 - e^{-x/\beta})^{k\alpha} \tag{3.13}$$

[1]Indeed, the approximate solution in this work requires only the first three moments of the applications' tasks, not the entire task distributions.

from which the pth percentile can be derived as,

$$x_p = -\beta \log \left(1 - (\frac{p}{100})^{\frac{1}{k\alpha}}\right). \tag{3.14}$$

Eqs. (3.12) and (3.14) show that the pth percentile, i.e., tail latency, of the target application can be expressed as a function of α_j's and β_j's and in turn a function of means and variances of the target task response times, according to Eqs. (4.2) and (4.3). Clearly, this establishes a link between job-level SLO, i.e., the job tail latency requirement, and task budgets.

3.3 Experiments and Results

In this section, we validate the predicted tail latencies, e.g., the 99th percentile of response times, for a target application in a system with a mixture of applications against those from the simulation. The accuracy of the prediction is measured by the relative error between the predicted value from the proposed model, t_{pred}, and the simulated one, t_{sim},

$$err = 100 \cdot \frac{t_{\text{pred}} - t_{\text{sim}}}{t_{\text{sim}}}$$

To illustrate the effectiveness of the proposed model, here we consider some typical scenarios with two classes of applications, a target application, which needs to be kept track, and a background application, which represents the remaining applications running in the system. The validation is performed under different settings for the target and background applications, including simple cases with the same type of service time distribution with different parameters and complicated cases with two different distributions. We incorporate both light-tailed and heavy-tailed distributions in the scenarios to represent different applications. In particular, we consider the following distributions:

– A light-tailed Exponential distribution (Exp) whose CDF is defined as [29]

$$F_{X_i}(x) = 1 - e^{-\frac{x}{\mu}} \quad x \geq 0, \tag{3.15}$$

where μ is the mean service time ($\mu > 0$),

– A heavy-tailed truncated Pareto distribution (TPa) [29] whose CDF is given by

$$F_{X_i}(x) = \frac{1-(L/x)^{\alpha}}{1-(L/H)^{\alpha}}, \tag{3.16}$$

where α is a shape parameter; L and H are lower and upper bounds, respectively.

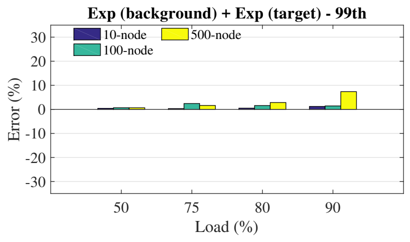
We report the results for the systems with different numbers of Fork nodes, i.e., $N = 10$, 100, and 500, and different weights for the target application in the mixture, i.e., 10%, 50%, and 90%, assuming that the tasks spawned from all the incoming jobs are randomly dispatched to N nodes in the system, i.e., $k = N$. Specifically, we consider three scenarios:

– **Scenario 1**–The same distribution (Fig. 3.2): Exponential distribution for both target and background applications with different mean service times, $\mu_{\text{tg}} = 13.78ms$ for the target and $\mu_{\text{bg}} = 4.22ms$ for the background.

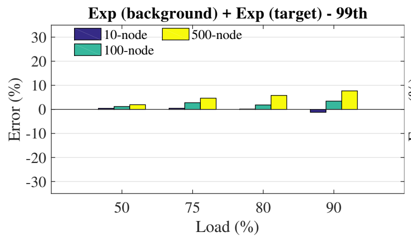
– **Scenario 2**–The same distribution (Fig. 3.3): truncated Pareto distribution for both target and background applications with the same coefficient of variation $CV = 1.2$ and different mean service times, $\mu_{\text{tg}} = 4.22ms$ for the target and $\mu_{\text{bg}} = 15.0ms$ for the background, which results in the distribution parameters $\alpha_{\text{tg}} = 2.0119, L_{\text{tg}} = 2.14ms, H_{\text{tg}} = 276.63ms$ for the target and $\alpha_{\text{bg}} = 2.0119, L_{\text{bg}} = 7.75ms, H_{\text{bg}} = 276.63ms$ for the background.

– **Scenario 3**–Different distributions (Fig. 3.4): Exponential distribution with mean service time $\mu_{\text{tg}} = 4.22ms$ for the target application and truncated Pareto distribution with coefficient of variation $CV = 1.2$ and mean service times $\mu_{\text{bg}} = 15.0ms$ for the background application.

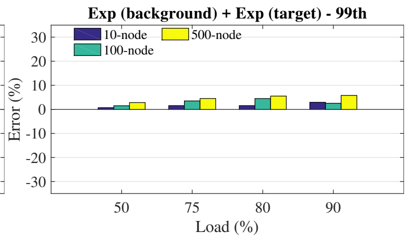
Figs. 3.2–3.4 show the prediction errors for all the cases studied with different load regions, i.e., 50%, 75%, 80%, and 90%. The results show that the prediction model is able to yield quite accurate predictions for the 99th percentiles of job response times, with most of the errors within 10% at the load of 75% or higher. This makes the proposed solution a



(a) Target = 10%

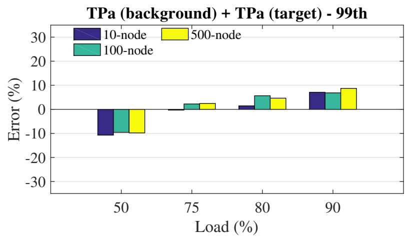


(b) Target = 50%

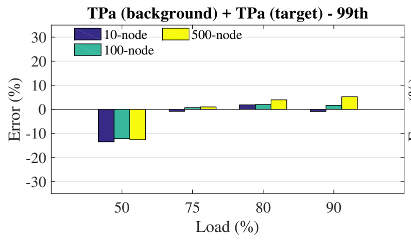


(c) Target = 90%

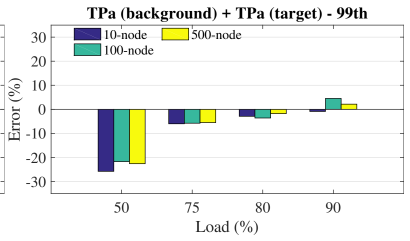
Figure 3.2: Prediction errors for the cases of different exponential distributions for both background and target tasks.



(a) Target = 10%



(b) Target = 50%



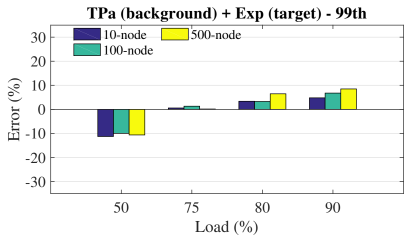
(c) Target = 90%

Figure 3.3: Prediction errors for the cases of different truncated Pareto distributions for both background and target tasks.

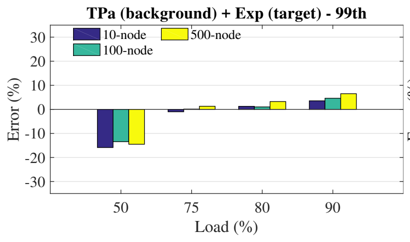
powerful tool for resource provisioning in datacenters with consolidated applications at high load regions, where precise resource provisioning is most desirable.

3.4 Related Work

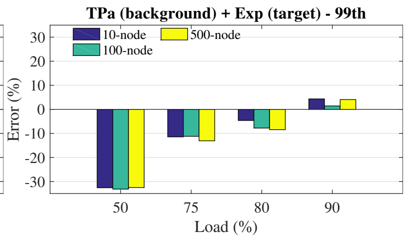
Fork-Join structures underlay many datacenter applications, in which the workflows are usually handled by a large number of nodes and the partial results from those nodes are then merged. Fork-Join structures are traditionally modeled by Fork-Join queuing networks (FJQNs) [8]. FJQN models have been studied extensively in the literature. To date, the exact solution exists for a two-node network only [35, 9]. Most of previous research efforts mainly focus on the approximation of job mean response time [9, 10, 36] and its bounds [37, 38, 43]. Some works attempt to find the approximation of response time distribution



(a) Target = 10%



(b) Target = 50%



(c) Target = 90%

Figure 3.4: Prediction errors for the cases of truncated Pareto distribution for background tasks and exponential distribution for target tasks.

for the networks applying simple queuing models for each Fork node, e.g., M/M/1 [42] or M/M/k [11], i.e., Poisson arrival process and exponential service times with one or more servers. For general service time distribution, the hybrid approach, which combines analysis and simulation, are usually used to derive the approximation for job mean response time [9, 12].

The approximation of tail latency for homogeneous FJQNs with phase-type service time distributions is introduced in a recent work [13], which is based on the analytical results from single-node and two-node networks. Unfortunately, the computational complexity of this approach renders it inapplicable to online resource provisioning. Also, it only covers a limited design space and thus cannot be used to facilitate resource provisioning in practice. In [52, 53], the authors propose to use a black-box approach for the approximation of tail latency at high load regions, in which each Fork node was treated as a black-box. The approach requires only measured means and variances of response times at the Fork nodes as input.

To our knowledge, no previous research effort provides analytical solutions for the approximation of the tail latency in FJQNs with consolidated workloads. The work in [53] does consider consolidated workloads but using the black-box approach, i.e., approximating task response time distributions based on the measured means and variances of response times on the Fork nodes. Here we present an analytical solution, i.e., a white-box approach, for the prediction of the tail latency in a consolidated environment and elaborate more on the consolidated impact on the accuracy of the prediction model. The proposed solution could be of great help for future research works that target the consolidated applications in FJQNs.

3.5 Conclusions and Future Work

In this chapter, we presented an analytical solution accompanied by several case studies for the prediction of tail latency regarding the consolidated applications in datacenter environments. Provided the results of three scenarios with different load characteristics, the prediction model has proven to be reliable, yielding all the errors well within the acceptable window of accuracy, 10% at the load of 75% or higher, for all the cases studied. These encouraging results suggest that the proposed model could be used to translate user demands into task budgets which in turn would help a scheduler make better scheduling decisions to satisfy user requirements. We look forward to extending this work for more sophisticated cases that include multiple processing stages, which is commonly in many production environments [58].

CHAPTER 4

Heterogeneous FJQNs

The unacceptable performance of many *user-facing* and *online data-intensive (OLDI)* services, at commercial datacenters *(DCs)*, revealed an emerging desire to developing smart, lightweight, and fast scheduling algorithms. This unveiled interest dictates a requirement to have an additional module devoted to predicting requests tail latencies. Where (1) applications tail SLOs translated into budgets, and (2) used to compare and secure proper resources ahead of time. Principally, this dire need to predict long tails out of distributions of latencies continues due to the sheer amount, and dynamic workload demands result in high variability within task *response times (RTs).* Nowadays, many applications requests poured into common heterogeneous infrastructure shared among applications. Therefore, heterogeneity in resources is also known to be one of the leading causes of the high variability among task response times. To this point, as a reason to mitigate the high latencies and to improve the decision-making at scheduler(s), many earlier scheduling algorithms and abundant of engineered techniques have been introduced [51, 59, 60]. Unfortunately, most of these scheduling designs, if not all, were based on a homogeneous assumption which renders their inability for adoption as practical solutions for heterogeneous environments. As a result, it becomes one of the principal causes of the expected poor user experience for all proposed solutions.

Consolidating cloud ecosystems helps reduce cost and improve the return on investment by sharing clusters resources at datacenters. Enforcing such practice would increase resources utilizations, but unfortunately, it comes with its pitfalls. In one hand, it introduced a significant depart from using organization-specific clusters. For example, many machines with various generational specifications and computing capacities are naturally used to satisfy the diversity in applications demands. At the other hand, consolidated applications typically need to run their daemons or other software components in the backgrounds of computing units. Sometimes, executing these software components lead to an increased amount of different processes hiccups at DCs. Hiccups differ concerning their durations, or the additional hardware characteristics might be required by daemons to operate. Further, contention over resources, either between applications or among the same application requests, maintenance activities, hotspots, energy management, thermal and power effects, all these mentioned pitfalls contributed to the sum up result in creating various inhomogeneous conditions [47, 61, 5]. Therefore, when non-homogeneous situations exist, providing low latency services at scale considered of at most important to many service providers.

The emerging dilemma challenged lots of researchers to offer better ways to keep the tail of latency distribution as short as possible for latency-sensitive services [49, 62, 63, 64, 65]. So far, none of the presented works were thoughtfully designed to take into account users' wishes to maintain the promised guaranteed services. The leading cause is due to the missing link between the system level requirements and the subsystem level performances. Fortunately, we presented in previous publications a prediction model [53], which serve to compensate for the missing baseline connection. The model has the potential to work as a universal answer for all types of DCs infrastructures, either shared or not [1, 66].

The design space for creating the before-mentioned sophisticated and diverse DCs' climates, at first glimpse, seems endless and very challenging. Still, it is possible to establish simplified models for the study of this research. For example, fork-join structures used generally to express the basic functional system at DCs, as they have become the core foundation for such environments. Fork-join structures traditionally modeled by a class of queuing network models known as *fork-join queueing networks (FJQNs).* The focus, mainly, will be on enabling computing units independently to demonstrate distinct responses when servicing a received group of tasks. Hence, by creating such heterogeneous FJQNs behaviors, it is possible to mimic what happens in production environments. This can be easily done by providing several case-scenarios of inhomogeneous situations linked to many examples from reality.

However, the lack of establishing inhomogeneous conditions in all previously researched models, particularly regarding the prediction of tail latency has become the downside that reflected at all. Therefore, it is believed that the outcome of this work might pioneer future studies of tail prediction concerning inhomogeneous conditions. Thus, the prime goal is to demonstrate those complex HFJQNs using model-driven simulations. In addition, a mathematical explanation emphasizes the white-box approach which is closed-form solution for any model whose analytical expressions for the means and variances of task response times are available, and comparisons between the two methods of prediction: white and black-box, will be drawn. Finally, as a contributed part to this research, a mathematical analysis for the function of budget translating unit will be presented. This analysis will be used to describe how to translate requested SLOs requirements into tasks' budgets to reserve proper resources ahead of time.

As explained, outcomes of this research might become a real treasure, for researchers interested in doing further investigations, utilizing the fork-join queuing models related to subjects of parallel processing. The lack of having enough studies for tail prediction at inhomogeneous environments are the main reason that motivates this work we provide. There are three new novel ideas that contributed to this research in the late sections: (1) convergence point for reliable metrics, (2) resource usage predictability, and (3) budget translation; which render the uniqueness of this work as it takes further steps toward providing a practical, reliable, and budget guaranteed scheduling mechanisms in the foreseeable future. Therefore, our research work formulated as the following: In section 4.1, a general overview of the derived prediction model and a closed-form solution for any inhomogeneous white-box models, in general. In section 4.2, we present abstract definitions of how to model inhomogeneous conditions in fork-join structures and list all possible case-scenarios of the defined inhomogeneous types, and it will be followed by a discussion about the results of the experiments, in section 4.3. The convergence point for reliable metrics described in section 4.4. In section 4.5, our explained method for budget translation. Finally, the related work in section 4.6 and the future work and summary in section 4.7.

4.1 An Overview of Prediction Model

There is a recognized fact about the behavior of tasks flow, in parallel systems, that are waiting at lines for their turn to execute. Mainly, for any G/G/m queuing model that is highly loaded, it is observed that the accumulated tasks waiting times behave as if they were independent variables following a distribution. The waiting times are the result of the build-up variabilities in tasks' executions processed in front of the waiting lines and, obviously, in addition to the known differences in tasks arrivals. Further, it was noticed that the distribution of the waiting times at a given line converged to a memoryless exponential distribution. Following this convention, it posited that task sojourn times could be approximated to a distribution too. Hence, it is assumed that the resulted fluctuations from variability in service times are going to be alleviated, due to the impact of having long queues.

Consequently, on the basis of extreme value theory, requests response times can efficiently approximate, if applying central limit theorem. In the sense that the two moments of historically collected response times for a single line, i.e., mean and variance, can be used to predict tail latency of response times, using any suitable distribution. Having mentioned that, generalized exponential distribution was determined, that it can capture different behaviors for the arrived jobs requests, e.g., heavy-tailed and short-tailed behaviors with regard to response times [1, 66, 53, 52]. The presented results in [52], had shown that generalized exponential distribution outperformed exponential distribution in the matter of predicting tail latencies. In the following is the generalized exponential distribution function:

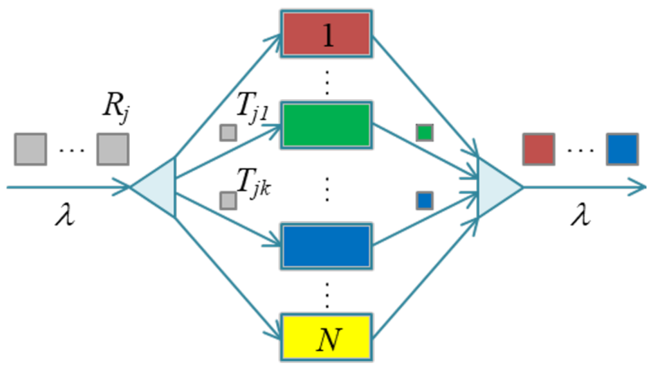


Figure 4.1: Simulation model for inhomogeneous scenarios, where incoming requests spawned to equal number of fork nodes at a cluster.

$$F_T(x) = (1 - e^{-x/\beta})^{\alpha}, \quad x > 0,\ \alpha > 0,\ \beta > 0, \tag{4.1}$$

distribution parameters α and β are shape and scale, respectively. The mean and variance of the task response time are given [57]

$$\mathbb{E}[T] = \beta[\psi(\alpha + 1) - \psi(1)], \tag{4.2}$$

$$\mathbb{V}[T] = \beta^2[\psi'(1) - \psi'(\alpha + 1)], \tag{4.3}$$

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

4.1.1 Prediction: Homogeneous

At a given fork-join structure, i.e., cluster of nodes, incoming users requests sent to a scheduler and forked to sub-requests/tasks that are mapped to the cluster's nodes. Now, assuming an approximated response times distribution at homogeneous condition, the yield is the derivation that can be used for tail prediction:

$$F_X(x) = \prod_{i=1}^{k} F_{T_i}(x), \tag{4.4}$$

$$x_p = F_X^{(k)^{-1}}(p/100)\,. \tag{4.5}$$

$$x_p = -\beta \log(1 - (\frac{p}{100})^{\frac{1}{k\alpha}}) \quad (4.6)$$

Where x_p is the pth percentile request response time; β and α parameters can be obtained from the measured mean and variance at a single node, eqs. (4.2) and (4.3). k is the number of forked tasks, $1 \leq k \leq N$-nodes.

4.1.2 Prediction: Inhomogeneous

The experiments carried out on all previous works was based on the homogeneous assumption. Many cases had been studied, even cases for consolidated applications were covered [53, 1]. However, none of the works was enough to demonstrate the prediction performance when tested under the non-homogeneous situations. Hence, putting the final piece of the puzzle into this ongoing research, a necessity to define ways to establish inhomogeneous conditions (i.e., HFJQNs) is direly needed for simulation experiments.

For the sake of clarity, consider a white-box approach and assume distinct distribution(s), either single or mixture of distributions, associated with each node in a given fork-join system. Assume only one of many conditions causes heterogeneity in the fork-join structure to be applied during the whole simulation period, and let G reference the condition type that could exist. Heterogeneous conditions in this research, generally, are driven by one of the followings: (1) node's distribution $f(x)$, (2) node's utilization ρ, and/or (3) the tail of distribution applied at each node τ.

Now, suppose there are N mixtures, of m distributions, at each of N nodes: SH_1, SH_2,…,SH_N. Let us assume the distributions resulted from combined effects of both **S**oftware and **H**ardware. It is possible that the mixture at each node/computing-unit might be uniquely different, i.e., $f_{j,i}(x) \neq f_{j,i+1}(x)$, for i = from 1 to N-nodes, as defined in followings:

- Service times x_j, for $j = 1, 2, \ldots, m$, follow $f_j(x)$ distribution, where $x_j \sim f_j(x)$ with μ_j and σ_j^2.
- Assume each distribution $f_j(x)$ at node i has weight p_j in the mixture SH_i, and assume that for a long run experiment, if used uniform distribution to maintain the weights $p_j's$, the resulted distributions are balanced. And the effective random distributions used to generate tasks service times:

$$X_i \sim f_i(x) = \sum_{j=1}^{m} p_{ij} \cdot f_{ij}(x). \tag{4.7}$$

The kth moment of X at any fork node i can be written as

$$\begin{aligned} \mathbb{E}[X_i^k] &= \sum_{j=1}^{m} p_{ij} \cdot \mathbb{E}[X_{ij}^k] \\ &= \sum_{j=1}^{m} p_{ij} \cdot \mu_{ij}^{(k)} = \mathbb{E}[\mathbb{E}[X_i^k | G_i]], \end{aligned} \tag{4.8}$$

where $\mu_j^{(k)}$ is the kth moment of X_j, and G refer to one of the previously defined types of heterogeneous conditions could exist at node i. Here, we assume only one for simplicity but in reality it could be more complex.

According to the law of total variance, the variance of X at fork node i is given by

$$\begin{aligned} \sigma_i^2 &= \mathbb{E}[Var(X_i|G_i)] + Var(\mathbb{E}[X_i|G_i]) \\ &= \sum_{j=1}^{m} p_{ij} \cdot \sigma_{ij}^2 + \sum_{j=1}^{m} p_{ij} \cdot \mu_{ij}^2 - \left(\sum_{j=1}^{m} p_{ij} \cdot \mu_{ij} \right)^2 \end{aligned} \tag{4.9}$$

Since tasks spawned to the subsystems came out from a large pool of many independent requests/applications, their *arrivals* at the system can be modeled using the Poisson process. Several studies have found the dispersion around the interarrival-times distribution is so small, i.e., *C.V.* = 1 to 2, which means using memoryless exponential is a reasonably acceptable model for simulating the arrivals of network traffics and datacenter applications [54, 67, 68]. Therefore, in this research, each Fork node viewed as an M/G/1 queue system, i.e., a Poisson arrival process with a general service time distribution and one service center.

The first and second moments of tasks waiting time at any Fork node, i.e., an M/G/1 queuing system using Takács recurrence theorem [26], are given as follows:

$$\mathbb{E}[W_i] = \frac{\lambda \mathbb{E}[X_i^2|G_i]}{2(1-\rho_i)} = \frac{\rho_i \mathbb{E}[X_i|G_i]}{1-\rho_i}\left(\frac{1+C_{X|G}^2}{2}\right), \tag{4.10}$$

$$\mathbb{E}[W_i^2] = 2\mathbb{E}[W_i]^2 + \frac{\lambda \mathbb{E}[X_i^3|G_i]}{3(1-\rho_i)}, \tag{4.11}$$

where $C_{X|G}^2 = \sigma_i^2/\mu_i^2$ is the squared coefficient of variation with σ_i^2 being the variance given by Eq. (4.9); and $\rho_i = \lambda \cdot \mu_i$ is the utilization or load on ith Fork node at average arrival rate λ.

Therefore, from task executions' mean Eq (4.8) and variance Eq (4.9), respectively, the mean and variance of response times of the target task at ith Fork node can be written as,

$$\mathbb{E}[T_i] = \mathbb{E}[W_i] + \mathbb{E}[X_i|G_i], \tag{4.12}$$

$$\begin{aligned} Var(T_i) &= Var(W_i) + Var(X|G), \\ &= (\mathbb{E}[W_i^2] - \mathbb{E}[W_i]^2) + (\mathbb{E}[X_i^2|G_i] - \mathbb{E}[X_i|G_i]^2), \end{aligned} \tag{4.13}$$

where $\mathbb{E}[X|G]$ and $\mathbb{E}[X^2|G]$ are the first and second moments of target task service times; and $\mathbb{E}[W]$ and $\mathbb{E}[W^2]$ are given in Eqs. (4.10) and (4.11). The remainder is straight forward using Eq. (4.5) and knowing the required moments for all of the N-nodes.

4.2 Definitions and Case-Scenarios

Thinking logically, any possible inhomogeneous condition, per node in a fork-join structure, mainly influenced by several elements. In this research, those elements categorized into two primary factors: software and hardware. Many real scenarios can represent the software side; for example, operating systems, resources management, queuing management, utilization level, frameworks, daemons, ..., etc. While the hardware capabilities can be represented by differences in computing power, memory speed, hard disk, I/O, ..., etc., both sides participate in shaping any inhomogeneous situation at DCs. Hence, it is presumed that in a DC, the noticed differences in computing capabilities among serving units are tangible. This is due to the influences of both resources hardware and competing's software, where their footprints on waiting times, queues, and tasks' executions are clearly witnessed. Therefore, from the perspective of a primary cluster component, i.e., computing units, an abstract description laid out to provide distinct realizations of how HFJQN could form. Each node behaves in a different way as if it has a distinct distribution associated with it, or as the collective mixture of distributions applied at each node is unique.

We will demonstrate the simulation experiments considering per node behavior (or a unique group of nodes) following unique distribution. Several defined prospectives used in modeling the heterogeneous simulations, where examples from real-world scenarios linked and used to explain some of the following abstract definitions. Therefore, experiments conducted assuming a cluster of: (1) different loads utilizations among nodes, (2) same utilization level for all nodes but different distributions' tails, (3) defined portions of homogeneous and inhomogeneous nodes [61][1], and finally (4) with a help borrowed from multicores and multiprocessing technologies, different architectural-design per node applied; e.g., different number of replicas, distribution type, mean service time, ...etc. For the later, as each node function independently from others, we speculate this case-scenario might become the key kernel for any future simulations targeting heterogeneous designs. Basically, it is done by allowing each node to be handled independently as a separate event-based simulation process, where a recipient fork node treats each encapsulated task arrived from the system level as a newly arrived job at the subsystem level.

[1]it is reported that 93% of google cluster is homogeneous which render the remaining as inhomogeneous.

4.2.1 Different load utilizations among nodes

Load imbalance among nodes might result from many arising factors. For example, it happens due to combined operational effects of unbalanced scheduling policies (e.g., random dispatching [51]) and/or running multi-degree of applications fanouts. Also, it might occur due to: existing various computational powers of computing machines, the prolonged effects of prioritization and resource reservations, or as a result of emerging hotspots, where particular items/entities become popular, and the requesting workloads for specific hotspots increase. Such unevenness in loads among a cluster's nodes could induce, in the long run, many distinguished levels of utilizations. Therefore, to establishing a similar scenario and composing HFJQNs rendering a clear difference in nodes utilizations, an exponential distribution selected for such case, but different mean service times put in use, where the max one selected to maintain the flow of the arriving requests into the system.

Provided several experiments with clusters consists of group(s) of 18 distinguished nodes, i.e., heterogeneous nodes, $h-$nodes $= 18$, where each one is uniquely utilized against others. Exponential distribution is the only applied distribution in this case, but with a different mean service time enforced at each node. Varied versions of utilizations for the primary selected mean service time $= 70ms$, which is the one utilized for the 90%, ends up creating a range of several mean service times for all provided N nodes at a given cluster. Experimented clusters are synthesized with the different loads utilizations, starting at 5% and ends with 90% utilizations and applying 5% incremental steps in between.

Following the white-box approach, table 4.1, we assume same distribution applied at all nodes, but different utilization (ρ_i) at each, where $i = 1$ to N nodes. Hence, estimating the needed two moments for all the N-nodes of fork-join structure can be easily done using Eqs. (4.13), and (4.12). The remainder is straight forward when using Eq. (4.5) with the required moments for the N-nodes. Prediction results presented in Figs. 4.2 and 4.3. Refer

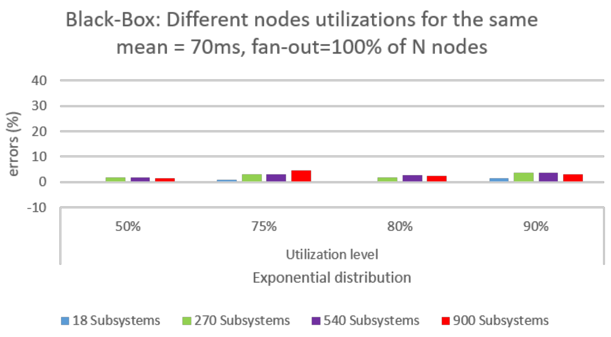


Figure 4.2: HFJQNs of different nodes utilizations, where the black-box approach used in solution, and the load level (here) determined by max node utilization.

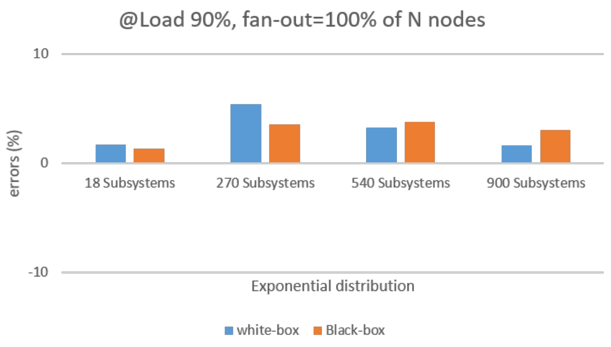


Figure 4.3: HFJQNs of different nodes utilizations, where the white and black-box methods prediction performances are compared, given the max node utilization is at load 90%.

Table 4.1: HFJQNs of different nodes utilizations

Parameters	$p_{ij} = 1$, $m = 1$, G = Utilization (ρ)
Arrival Distribution	Poisson, $E[X\|\rho = 90\%] = 70ms$
Service Distribution(s)	Exponential, $E[X_i\|\rho_i]$
Manipulated at $node_i$	$\rho_i =$ in $[5\%\ to\ 90\%]$, Step $\rho = 5\%$
# Heterogeneous Nodes	$h-$nodes = 18 (or multiples of $h-$nodes)
Cluster Sizes	18, 270, 540, 900

to discussions regarding all the experiments-results of this section as well others which will be explained with more details in section 4.

4.2.2 Variant distributions tails among nodes

At high load regions, where lots of incoming applications requests are in waiting lines, contention for shared resources mounts up. Not to mention the diverse computing powers of resources that could exists among nodes. This might exhibit distinct differences in the overall tail response times at each node. Therefore, the combined effects of both, i.e., the increasing amount of applications demands, and the noted varieties in cluster hardware, can be presented in such case-scenario where each node bounded by a different tail τ_i than other nodes. Consequently, to demonstrate the effectiveness of the prediction model, examples of HFJQNs using multiples of distributions with different tails, and each node is presumably associated with one of those several tailed-distributions.

In the following experiments, simulations with clusters consist of group(s) of nine distinct tails for nodes, i.e., heterogeneous nodes, $h-$nodes $= 9$, where each one has a unique distribution tail different than others. Heavy-tail Pareto distribution with the same mean service time, $\mu = 4.22ms$, and lower bound $L = 2.14ms$ is assumed for all. The array of the nine tails, table 4.3, originated by manipulating the shape parameter α_i, where they used to distinguish each node in a single group of nine tailed-nodes. When the cluster scale up, it will become multiple of the same single nine-nodes group.

Following the white-box approach, table 4.2, we assume same distribution applied at all nodes, the same utilization level as other nodes, but different tail (τ_i) at each, where $i = 1$ to N nodes. You can solve the complete FJ structure knowing the required moments for the N-nodes from using Eqs. (4.13), and (4.12) and then applying Eq. (4.5). Prediction results presented in Figs. 4.4 and 4.5. Discussions regarding all the experiments results of this section as well others will be explained with more details in section 4. .

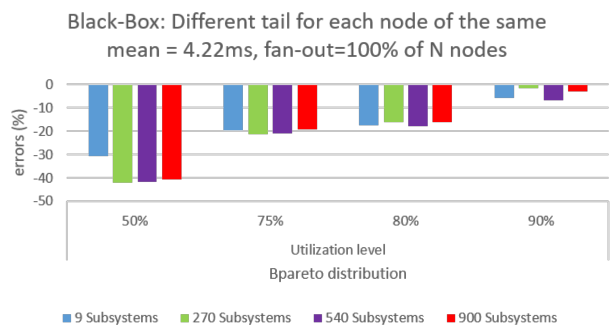


Figure 4.4: HFJQNs of nine distinct tails associated with all fork nodes, where the black-box method used in solution.

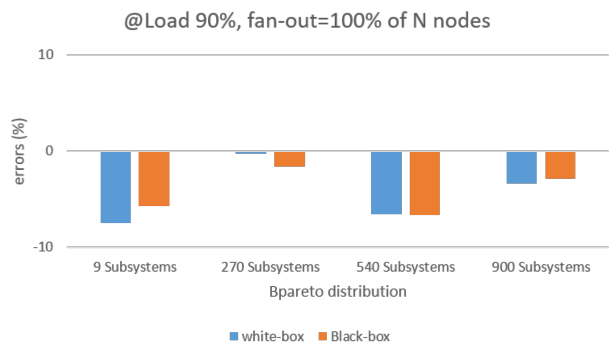


Figure 4.5: HFJQNs of nine distinct tails applied at all fork nodes, where the white and black-box methods prediction performances are compared at load 90%.

Table 4.2: HFJQNs of nine tails associated with all nodes

Parameters	$p_{ij} = 1$, $m = 1$, G = Tail (τ)
Arrival Distribution	Poisson, $E[X] = 4.22ms$
Service Distribution(s)	Truncated Pareto, $E[X_i \vert \tau_i]$
Manipulated at $node_i$	τ_i = see Table 4.3
# Heterogeneous Nodes	$h-$nodes = 9 (or multiples of $h-$nodes)
Cluster Sizes	9, 270, 540, 900

4.2.3 Separate portions of homogeneous and inhomogeneous nodes

A straggler is a well-known phenomenon that might temporarily occur when service providers scale up their systems to cope with the growing needs for extra resources. Even if a current state of a system kept intact, there would be a potential chance for one or

Table 4.3: Pareto: different tails (τ_i) for different shapes (α_i)

Distribution Tail τ_i	Shape Parameter α_i
503.53ms	2.0210
450.00ms	2.0200
402.65ms	2.0189
350.07ms	2.0173
301.53ms	2.0153
250.39ms	2.0123
200.59ms	2.0078
150.51ms	2.0000
100.09ms	1.9833

multiple serving units to slow down their executions of the incoming tasks. This might happens due to: (1) spikes in requests for memory access or in CPU activity during tasks processing, (2) interference, in accessing storage servers or from irrelative network traffic, (3) maintenance activities such as garbage collections, data reconstructions, or (4) energy management and so on. All could induce similar situations for having group of straggler computing stations existed at any given cluster. For example, it has been known from several studies on production traces of google cluster, that a realized significant division of the cluster was homogeneous while the remaining division was inhomogeneous [61]. This fact is the basis of the subsequent experiments.

Assuming, a 90% of the given cluster nodes associated with exponential distribution, for $\mu = 4.22ms$, and considered the homogeneous part. While the remaining nodes, i.e., 10%, left to be handled by Truncated Pareto distribution with max tail (upper bound) $= 503.530ms$. We will follow through this example setup, and in another experiment this ratio will be tuned as depicted in figure 4.8.

For the white-box approach, table 4.4, the same utilization level are enforced at all fork nodes. We assume two different distributions $f_i(x)$ associated with two groups of defined nodes' percentages, i.e., $h-$nodes $= 90\%$(homogeneous) $+ 10\%$(inhomogeneous), where $i = 1$

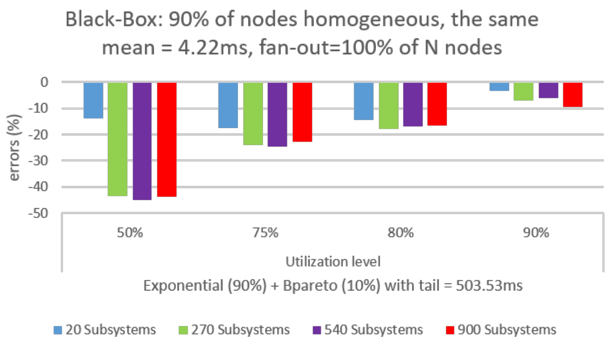


Figure 4.6: HFJQNs of applying 90% homogeneous (exponential) and 10% inhomogeneous (bounded pareto) for all fork nodes, where the black-box method used in solution.

to N nodes. For solving the Fork Join structure of N-nodes refer to Eqs. (4.13), and (4.12) to estimate the required moments for all N-nodes, and then use Eq. (4.5). Prediction results presented in Figs. 4.6 and 4.7. Discussions regarding all the experiments results of this section as well others will be explained with more details in section 4.

In a related scenario-experiments, explained in figure 4.8, we push this case a little further and observed the impact when tuning the portion of the inhomogeneous nodes. Thus, the changing effects on: jobs mean response times, the $99th$ percentile of the actual experiments response times, and the predicted $99th$ percentile tail-response times were studied, and the resulted outcomes presented as shown. Truncated Pareto distribution of the same tail (i.e., $503.53ms$), and exponential distribution with mean execution times $= 4.22ms$ were applied. The outcomes of this experiments are no different than the basic one, where the errors in prediction stay the same even knowing that the situation changed completely to homogeneous at the two ends of the figure, i.e., totally dominated by exponential distributions or Pareto distributions.

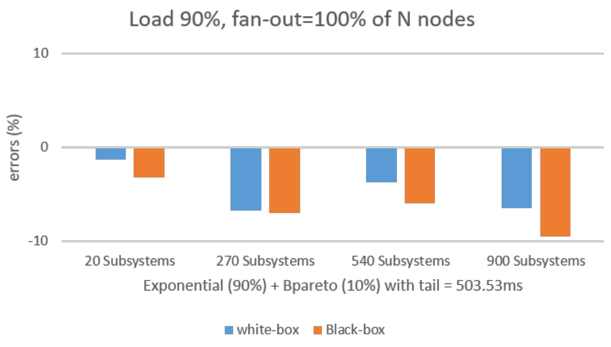


Figure 4.7: HFJQNs of applying 90% homogeneous (exponential) and 10% inhomogeneous (bounded pareto) for all fork nodes, where the white and black-box methods prediction performances are compared at load 90%.

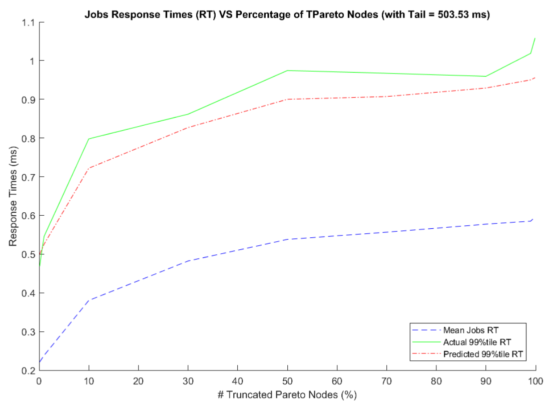


Figure 4.8: Jobs response times (RT) against increase percentage of inhomogeneous nodes (bounded pareto with tail = 503.53 ms, and total cluster size = 900 nodes).

Table 4.4: portions of homogeneous and inhomogeneous nodes

Parameters	$p_{ij} = 1$, $m = 1$, G = Distribution $f(x)$
Arrival Distribution	Poisson, $E[X] = 4.22ms$
Service Distribution(s)	TPareto & Exponential, $\mu = 4.22ms$
Manipulated at $node_i$	$f_i(x)$ = Expo or Pareto ($\tau = 503.53ms$)
# Heterogeneous Nodes	$h-$nodes = 90% Expo + 10% Pareto
Cluster Sizes	20, 270, 540, 900

4.2.4 Nested event-based simulations model for multicores technology

A proposed scenario for heterogeneous modeling is to deviate from usual practice in implementing event-based simulations and construct a simulation model of different architectural designs. For example, different nodes structures concerning: replicas, distributions, utilizations, and even for establishing huge gaps between mean executions times, all can apply at a single round if using such simulation models. Multicores technology and multiprocessing technique both were considered at the design stage to help exploit the suggested modeling scenario to the limits. Therefore, it became the sole motivation for conducting the following experiment, as it was desired to test the results of the prediction model when a situation where different nodes structures exist.

A vector of twenty-six distributions synthesized initially from nine distinct distributions, ten different mean service times, and nine Pareto distributions of different tails set with same mean service time, $\mu = 4.22ms$, are used for the experiment. The elements from the vector are assigned randomly to all nodes following uniform random distribution. A range of random replicas numbers is set arbitrary for each node. Thus, diverse architectural designs enforced per node. The Pareto distribution used to generate target requests based on a predefined percentage to the system scheduler, where tagged forked-tasks sent to subsystems. Each subsystem handled as different event-based simulation processes, where the tasks of the untagged job sent to null-event used to advance the sub-simulation time. All sub-simulations function independently with varying configurations of their networks, and the numbers of generated task requests at each node are set to be unlimited. The created task requests considered as the influence of having background applications involved. Recognizing the behaviors of background applications helped to study their impacts on target tasks received from the control system, i.e., from the parent simulation, see Fig 4.9.

4.3 Results and Discussion

To our knowledge, no one so far exploits the tail prediction at heterogeneous environments the way we did in this research work. Also, all the state-of-the-art prediction models heard of, are known for targeting only homogeneous FJQNs, which lead to no available model

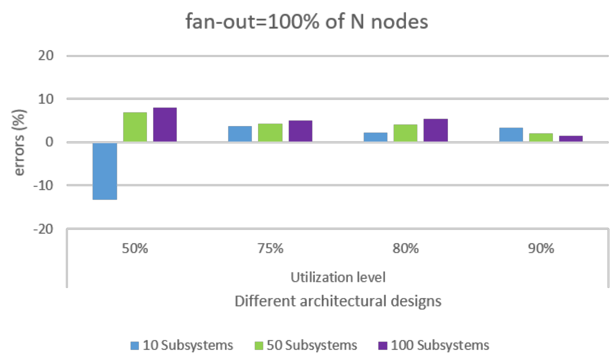


Figure 4.9: HFJQNs of applying different architectural designs at all fork nodes, where the black-box method used in solution..

for accuracy comparisons. Therefore, we used both approaches of our model: the white and the black- boxes methods, for comparing prediction results where it applicable.

For the white-box approach, to relax the complexities in computing the root, we even the number of nodes in all experiments groups, where the heterogeneous nodes of an experiment group, $h-$nodes, have different computing capabilities, e.g., unique distributions, or mean service times,. . . etc. Then it become multiples of the same $h-$nodes as the system scale out to large cluster sizes, e.g., 270, 540, and 900. Except the last case-scenario, where the $h-$nodes have different characteristics configurations per each scaled-out experiment, and that explains why only black-box is presented in that case.

Further, we numerically computed the root using the Levenberg Marquardt algorithm as the root-finding package comes by default with Matlab fsolve function. Applying the root-finding function with this algorithm requires to assume an initial value to begin with. From observations, it is found that this initial value has its impact and at the same time dictate the accuracy of tail prediction results. Hence, we conducted a small research to observe the proper range of initial values for all heterogeneous conditions, to determine one common range that would work under the selected root-finding package. From our analyses, we find that the right initial value for the utilization experiments using light-tailed distribution (e.g., exponential) is any value (r_{init}) that fell within the range: $0.99 \leqslant r_{init} \leqslant 3.05$. Also, for the remaining tests types, either applying the heavy-tailed Pareto of different tails or in combination with other distributions, the best initial value is: $0.24 \leqslant r_{init} \leqslant 0.47$. The later range works well too with the hetrogeneous cases covered in another research work conducted in parallel with this one, where mixtures of distributions applied at each node. Actually, it is not clear if this observation of range effect is due to the Matlab root-finding package or the estimation method which needs another research outside the scope of current study.

For the types of HFJQNs in section 3, except the last, Figs. 4.2 −4.7 present the prediction errors for the 99th percentile response times at loads: 50%, 75%, 80%, and 90%, for all HFJQNs of N's nodes that are running single serving unit at each. The odd figures, i.e., 4.3, 4.5, and 4.7, present comparisons of prediction errors between the Black-box HFJQNs and the White-box HFJQNs at load 90% only. The model yields quite accurate approximations for tail latency at high load region, less than 10% and 20% at the load of 90% and 80%, respectively. Further, in Fig. 4.2, for all the cases with the exponential distribution, i.e., different nodes' utilizations, the model gives accurate predictions across the multiple HFJQNs sizes studied. This is consistent with earlier observations in [53], i.e., the lighter the tail, the smaller the prediction errors. The results shown in the odd figures, i.e., the outcomes of comparisons between white-box and black-box, ideally, should be equivalent. The differences are introduced due to simulation and measurement errors, i.e., based on 95% confidence. For the example case with the nested simulations utilizing multi-cores technology in subsection 3.4, the current implementation code rely heavily on memory where the experiments limited to small cluster sizes, i.e., 10, 50, and 100, and that is due to the limited resources of using single core7-CPU testing PC. Fig. 4.9, shows the prediction errors at the determined load levels, which to our surprise the prediction errors are reduced, with less than 10% mostly across all load regions, and less than 5% for loads at 75% or above. This further asserting our postulation about the reliability of using this model assuming no advance knowledge of nodes distributions, and even under such heterogeneous scenarios where huge differences in performance applied among nodes architectures.

Lastly, **one of the main questions that haven't been answered yet is the considerable difference in prediction errors between heavy and light-tailed workload distributions?** It seems the model successfully predicts the light-tailed within the expected window of errors across all load-levels. At the same time, it underestimated the heavy-tailed distributions at low loads, and once it reaches high loads, it overestimated even though it stayed within the expected window.

As known, there are two competing effects in the approximation: 1) the impact of using extreme value theory, and, 2) modeling our solution using a generalized exponential distribution. First, the solution is not an exact estimate but approximated; we use extreme value theory, which predicts the worst-case scenario. That explains why there is an expected small percentage of errors added to the approximation. Second, the basic concept is that at high load regions, due to queuing effect, the distribution of task response times, of a single forked-node, can be modeled as exponential as its CV becomes near to exponential in its behavior/characteristics, i.e., CV in the range $[1 - 1.2]$. Third, the used generalized exponential model is inherently exponential, where the long tail can be considered as an added feature. Therefore, it will not have trouble predicting distributions of short tailed-family at most load regions as we had seen with the Weibull and the Exponential distributions, chapter 2. However, for heavy tail distributions, and based on the concept, at high load regions, when queuing effects dominate the existed workload distribution, the 10% errors in prediction observed in all applied scenarios of workload distributions, is considered enough for the model to satisfy the argument.

Then why is there an increase in error if we increase the cluster size? To provide more emphasis on this, it is known that based on the previous observations, there will be high variance among queues lengths at all forked-nodes, i.e., increased chances of having the head of the line blocking phenomena, where at least in one queue node, several long consecutive tasks could mount-up after each one, which cause underutilization for the system as a whole. Once we monitor the entire FJ system and take utilization-snapshots at every moment, it will become apparent the high variance of queues lengths as well as the tasks service-times among nodes-queues, and even though, in the long run, the load (ρ) level maintained fixed for each queuing node. Hence, the result of this momentary underutilization in the FJ system is response times that are dispersed and clustered to form a distribution. Consequently, the tail of this new converged response time distribution becomes longer due to the high variance in jobs-responses, which escalate as we grow the forked-nodes size in par with the forked-tasks amount, even though the result from prediction stays within the 10% error window.

4.4 Convergence Point for Reliable Metrics

The two performance metrics are essential for the work of the prediction model, and both are used to interpret any node's ability to serve a demanded latency response. Historical collections of task completion times help in inferring the two metrics and determining the resource capabilities as well. Therefore to gain better prediction, a crucial point is to learn the least amounts of archived responses, which is necessary for assessing the reliability of the metrics' measurements at a node. For example, assuming homogeneous conditions, it is known from the budget translation section that a single round of translation for a provided target response helps achieve the required nodes' measurements. Hence, μ and σ^2 can be obtained when numerically determine the value of C ratio[2], and infer both: α and β. Now,

[2]In contrast to budget estimation, calculating C ratio is not restricted by selecting a value that causes non-negative bounds, where C ratio could be less than 1. In some published analysis [54, 67, 68], it is realized that the $1 < CV \leq 1.2$, which means CV = 1 + Const. Equivalently, when taking advantage of the reciprocal relationship, C ratio = 1 - $\overline{\text{Const}}$. Based on our observations of measuring the right $\overline{\text{Const}}$, the

assuming a 10% window of errors in prediction is acceptable and a required 95% confidence level. Chebyshev's inequality can apply besides the statistical metrics gathered from clusters nodes.

$$P(|X-\mu| > \varepsilon) \leqslant (1 - confidence) \tag{4.14}$$

Where, ε is the accepted error for mean of task responses, i.e., $\varepsilon = (\textit{Accepted Error Ratio} \cdot \mu)$. Now, assuming collected samples are independent, *central limit theorem* can be used for further derivation to normalize the $\sigma^2_{\bar{x}}$ of the population distribution into σ^2/n for collected samples, as the following:

$$\frac{\sigma^2_{\bar{x}}}{\varepsilon^2} = \frac{\sigma^2}{n \cdot \varepsilon^2} \tag{4.15}$$

Therefore, we have,

$$P(|X-\mu| > \varepsilon) \leqslant \frac{\sigma^2}{n \cdot \varepsilon^2} \tag{4.16}$$

And from both Eqs. (4.14) and (4.16),

$$n = \frac{\sigma^2}{\varepsilon^2 \cdot (1 - confidence)} \tag{4.17}$$

Fig. 4.10, demonstrate an example where the parameters empirically gathered from carrying on two different types of experiments used for the estimation. The obtained statistics are from conducted homogeneous experiments [1], where mean and variance in responses observed only for tagged tasks at workers. The used clusters were of different sizes (e.g., 10, 100, and 500 workers), and the collected responses of target tasks were based on predetermined target percentages (e.g., 10%, 50%, and 90%). Hence, the influence of having

value found to be the ratio between mean execution time over the mean response time is the best fit value. Both parts of the ratio easily obtained from user's required tail latency and its ratio to mean execution time. The ratio (r) between target tail response time and mean execution time according to the determined load level (ρ) as in $\rho = 1 + r^{-1} \ln(1 - (\frac{p}{100})^{\frac{1}{N}})$, more in next chapter.

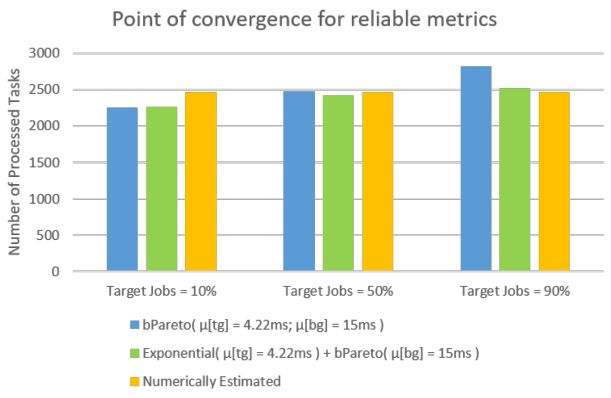


Figure 4.10: Samples amount required for reliable metrics, check scenarios: 2&3 in [1].

background applications was involved in the experiments. In the first scenario, the background was of the same distribution but applying different mean service time than the target, and the second one, the background was of different distribution. The result in the figure shows that if a particular application gets to dominate a large percentage of the entire workload, the number of needed samples slightly increases $\leq 20\%$, where both two ends stayed within the expected 10% of target if set as a reference point when it dominates half of the workload. That is, once considered the addition of this slight amount, it doesn't matter how significant the portion of a target application will be. We follow the above derivation to mathematically estimate the required amount of samples for the same target tail once translated into μ and σ^2. The results found to be about the same within an accuracy of more than 87.5% compared with the one estimated from measurements. It worth to mention that the observed differences in the introduced errors caused by the average μ's might be due to simulation and measurement errors, which could also negatively play a role in the changed portions of the target shown in the figure.

Further, the amount of the required samples can be tuned for a less amount once the accepted error ratio of the mean latency increased from the 10% shown in the figure to a little higher (e.g., 20%). Therefore, this method is a very crucial tool as it presents a way to estimate the required amount of sample response times for getting a reliable *mean latency* measurements of task processing at a fork node. Finally, using this approximation approach for estimating the required sample size is unique in a way that it doesn't need to know the distributions or nodes' statistical performances beforehand [69]. At the same time, it doesn't require to use approximations, which incurred a high margin of errors when neither distributions nor nodes' measurements are known [70, 71].

4.5 Budget Translation

One of the unique and potential benefits using the prediction model is the ability to translate any required latency into task performance budgets, e.g., μ and σ^2. Consequently, from the given allowed budget, any landed job request negotiates for a range of nodes, based on the knowledge of subsystems performance metrics, that could serve while guaranteeing SLO requirement satisfaction. For that reason, a question of how to work out the budget translation using this model should arise. From Eqs. (4.2), (4.3) and (4.5) already have the following:

$$\mu_T = g(\alpha, \beta) \tag{4.18}$$

$$\sigma_T = h(\alpha, \beta) \tag{4.19}$$

$$x_p = f(\alpha, \beta, p, k) \tag{4.20}$$

Where μ_T and σ_T are the mean and standard deviation of task responses, and x_p is the required tail in response times. Now, assume ratio $C = \mu_T/\sigma_T$, where $C \geqslant 1$. This provides us with four equations and four variables We can numerically solve it, and hence, the translated budget we will be:

$$B_T = \mu_T \pm \sigma_T \tag{4.21}$$

Table 4.5: Translated budget using two different C ratios

K	Latency	C	μ_{SLO}	σ_{SLO}	Upper limit$_{SLO}$	Lower limit$_{SLO}$
20	100ms	1	13.16ms	13.16ms	26.32ms	0
	100ms	15	60.46ms	7.22ms	67.68ms	53.23ms

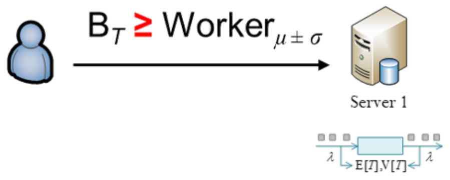


Figure 4.11: Using translated budget to determine the right destination.

For a demonstration purpose, assume homogeneous condition with the given user requirements: 99%ile, $x_p = 100ms$, and amount of forked tasks $k = 20$. Having applied the above method led to the results shown in table 4.5, where the translated budget can be used to direct tasks as in Fig. 4.11. Choosing a C ratio ≈ 1 sounds reasonable to attain the performance measurements of a single node. A core principle of this model approximation is that at high load region, waiting times for a single node follow an exponential distribution, which means μ and $\sigma = 1/\lambda$. Therefore, it is safe to assume for budget translation, setting C ratio $= 1$ would provide us with approximately the same μ expected at any single node.

Accordingly, what happens if the C ratio changed to different values? For example, assume changing C ratio takes the range between 1 to ∞. Figures 4.12 and 4.13 demonstrate the impact of tuning C ratio at the translated budget. Note that the observed changes in the boundaries stop when $C = 15$, but the increase in μ and the decrease in σ values continue to untile $C = 31$. This means, working different C ratios helps relax (low C) or tighten (high C) the amount of provisioned resources for the allowed budget.

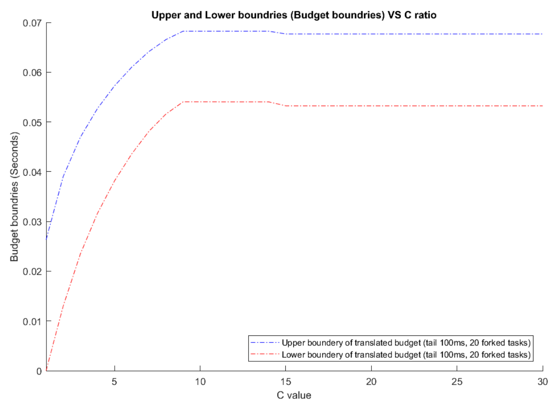


Figure 4.12: The two boundaries of translated budget.

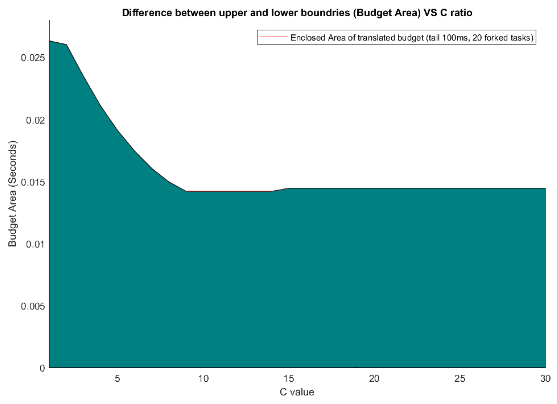


Figure 4.13: Translated Budget area: the enclosed area between the two boundaries.

4.6 Related Work

Fork-Join structures underlay many big data applications, whose execution usually are handled by a large number of nodes and the partial results from those nodes are then merged. Fork-Join structures are traditionally modeled by Fork-Join queuing networks (FJQNs) [8]. FJQN models have been studied extensively in the literature. To date, the exact solution exists for a two-node network only [35, 9]. Most of previous research efforts mainly focus on approximating mean response time for FJQN model [9, 10, 36] and its bounds [37, 38, 43]. Some works attempt to find the approximation of response time distribution for the networks applying simple queuing models for each node, e.g., M/M/1 [42] or M/M/k [11], i.e., Poisson arrival process and exponential service times with one or more servers. For general service time distribution, the combination of analysis and simulation are usually employed to derive the approximation for mean response time [9, 72, 12].

The approximation of tail latency for homogeneous FJQNs with phase-type service time distributions was introduced in a recent work [13], which is based on the analytical results from single-node and two-node networks. In [52, 53], the authors proposed to use black-box approach for the approximation of tail latency at high load regions, treating each processing node as a black-box. The approach requires only means and variances of response times at the processing nodes as input. However, it was based on the homogeneous assumption. Therefore, in this research analytical approach derived for use in the introduced inhomogeneous FJQNs.

4.7 Future Work and Conclusion

So far, provided there is a clear distinction between the inhomogeneous and homogeneous environments, the original-derived prediction model still holds. At schedulers, if decision-makings established only based on queues lengths, it could sometimes be deceiving [51]. Henceforth, engaging the knowledge of the computing capabilities for cluster nodes to derive the resources allocation is more reasonable for gaining reliable performance. Schedulers would become able to define budget boundaries for guaranteed users' satisfaction. Based on the performance measurements feedbacks, dispatching systems can specify ranges of nodes that could afford to serve users' requests within the given SLOs requirements. A good example is to perceive an experienced taxi-driver who takes advantage of the knowledge he has about destination. And he uses the knowledge to ride his vehicle and steer the wheel toward their proper addresses. This is a crucial improvement that we could get from using the prediction model, which seems already lacked to exist at other models.

In this chapter, while assuming heterogeneous conditions existed, we provided several study cases regarding the prediction of tail latency. Many conditioned-scenarios were covered, where different distributions and unique characteristics considered for the demonstration of cluster nodes. The model has proved to be reliable as it performed well within an acceptable window of errors for all the studied cases. The encouraging outcomes stand for adopting this prediction model in designed schedulers for improved decision-making and providing guaranteed services. Furthermore, it is expected that this model could fits rightly in more sophisticated scenarios, where multi-stage scheduling is acquired [58]. Therefore, extending the use of the prediction model is the next future work, as the design nature of many production environments requires to include more processing stages instead of only one fork-join phase.

CHAPTER 5

Achievable Cluster Utilization for OLDI applications

Scale-out applications are on the track of becoming the norm in users' everyday life. The use of such applications seems to be extending in many internet services that we tend to use heavily. For instance, keywords-searching, social-networking, user-facing services of big data analysis, and so on make it essential to realize the significant impact of the kind of applications that are referred [to] in this context, in users' daily works. More demanding is that these particular-types, known as online data-intensive (OLDI) services, usually require rigorous response times toward the satisfaction of end-users. For example, OLDI-services jobs may fan out a large number of tasks, where they processed at different computing units in cluster(s) at a data-center. At the same point, they have to meet stringent tail-latency Service Level Objectives (SLOs). Therefore, service providers usually resort to allocating more resources to such applications without following clear guidance about how much needed for serving a required tail SLO.

Usually, this approach of allocating resources often leads to data-center resource over-provisioning and result in high-cost excised from wasted-resources. Also, not forgetting to mention that it generally causes resources to run at low loads. For example, aggregate CPU/memory utilizations in a Google cluster with 12,000 servers [5] and a Twitter cluster with thousands of servers [6] are mostly less than 25%/50% and 20%/40%, respectively. However, and again, due to various demands of the arriving stream of diverse job-requests, where all are competing for cluster resources, and a possible high variance that might exist within the DC workload, the desired tail latency would become difficult to contain without throwing a significant amount of resources to overcome the problem. There is no way to interpret the current malpractice of handling clusters-resources into something else except a lack of having a good understanding of the issue at hand.

Therefore, this unveiled-absence of having a good comprehension of exactly how much resources should be overprovisioned and to what degree? was the primary motivation of our research work. Thus, we derived an approximation tool for estimating the performance upper bound of a utilization level, ρ, that can be achieved at a given data center. This estimate is based on assumption of a relationship for a selected target-ratio between the desired tail latency SLO of response times and the average service time, i.e., *tail-to-mean-ratio (r)*. Hence, the proposed tool can be used for characterizing the upper bound performance of the achievable utilization level.

However, as an application of our proposed solution, chapters [2, 3, 4], the model is applied to the prediction of achievable resource utilization under any given tail-latency SLO. First, as mentioned, we will provide a derivation for characterizing the performance upper bound for a particular class of OLDI services. Then we will use our model solution to validate the derivation and, at the same time, use both: the solution-model and the derivation to predict achievable resource utilization under any given tail-latency SLO. Also, to push the barrier a little further, we will use two extremely workload scenarios with respect to workload variance and study their effects on the utilization prediction.

Lastly, handing out this approximation tool at the hand of datacenters service providers would allow them to overcome many obstacles stemmed from using that lame old-style where they rely on blind-overprovisioning of clusters-resources. The remaining sections of this research formulated as the followings: In section 5.1, a general overview of the derived upper-bound performance model. In section 5.2, we present an analysis of our model assuming best case scenario of workload conditions. And, in section 5.3, we validate our model comparing its results with simulations and, numerically - using the GE-prediction model. Also, it will be followed by a discussion about possible ways that could be used to improve the utilization,

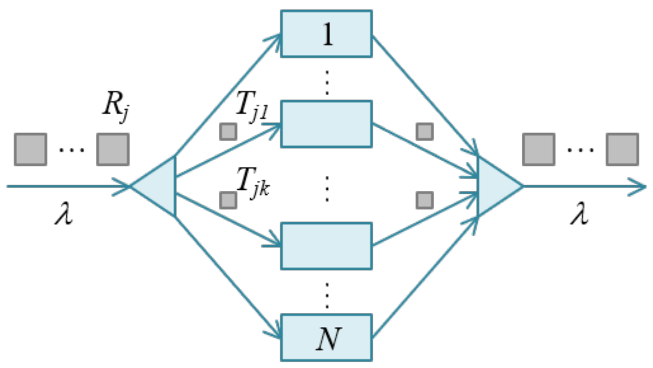


Figure 5.1: Fork-join process: a job request R arrives at a task dispatcher which spawn and distributes N tasks to N workers.

in section 5.4. Finally, the related work in section 5.5 and the future work and summary in section 5.6.

5.1 Model and Solution

In the field of parallel computing systems, it is recognized as a fact that the behavior of tasks flow, can be generally modeled as a Fork-Join Queuing Network (FJQN), see figure 5.1, which represent the job execution process for most of OLDI services. Each arriving job forked to several tasks that are dispatched to different fork-nodes and they wait at lines for their turn to execute. Jobs arrivals can be modeled following a distribution process, and the same goes for tasks executions, and both distributions are either similar or different. The finished tasks of a job are merged and the last processed task will dictate that job response time.

For this approximation to work, it is assumed an M/M/1 Fork-Join-queuing-network model design, and the worst case of fanout numbers that equal to the Fork nodes. The basic concept behind these two assumptions is that if we considered the best-case scenario of low variance among workloads that leads to estimate the maximum barrier of achievable utilization, ρ, we can easily derive a model to predict the achievable upper bound utilization for other conditions, where a high variance in system workload do exist. It is already known for a fact that the usually encountered workloads, in reality, have high variance (i.e., C.V > 1), which is due to the mix between short and long jobs at the waiting lines of processing nodes. Several studies have found the dispersion around the interarrival-times distribution is so small, i.e., C.V $= 1$ to 2, which means using memoryless exponential is a reasonably acceptable model for simulating the arrivals of network tracs and datacenter applications [54, 67, 68]. This selection become clear since tasks spawned to the subsystems assumed that they came out from a large pool of many independent requests/applications. Hence, due to its coefficient of variation (i.e., C.V $= 1$), using the exponential distribution for simulating cluster-workloads makes it the best candidate to serve as modeling component of our solution for both the task interarrival time distribution and the service time distribution at each queuing server. This means that both distributions are found to be shorter tailed than the measured statistics for datacenter applications with coeffeciant of variation C.V > 1 in general [30], including the Google search service [54, 30]. As a result, due to low variance one can expect that the solution to the current model would lead to load upper bounds as a function of the tail-latency SLO.

Next, we establish a link between ρ and $\{r, p\}$. We first note that given the cumulative distribution function for normalized job response time t/S, $G(t/S, \rho, N)$, which is load and fanout degree dependent, we have,

$$\frac{p}{100} = G(T/S, \rho, N) = G(r, \rho, N) \tag{5.1}$$

This expression establishes the relationship between the cluster load ρ and the tail-latency SLO tuple, $\{r, p\}$, at any N. So the key is to derive $G(t/S, \rho, N)$ for the above Fork-Join queuing network model.

To this end, we first note that the cumulative distribution function for the task response time at each M/M/1 Fork queuing server, $F(t)$, is known [73] and can be expressed as follows,

$$F(t) = 1 - e^{-(1-\rho)t/S} \tag{5.2}$$

Now assuming that the task response times for tasks at different queuing servers are independent of one another, we have, according to the extreme value theorem [74],

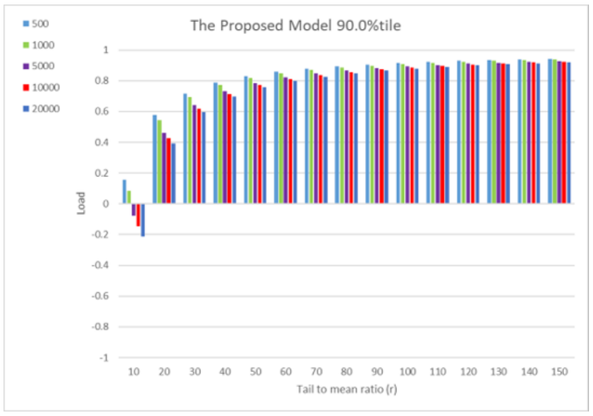
$$G(t/S, \rho, N) = F(t)^N = (1 - e^{-(1-\rho)t/S})^N \tag{5.3}$$

Interestingly, it can be easily shown that the above solution coincides with the black-box solution given in [53] in the case of the FJQN with M/M/1 Fork nodes. The key difference is that the current approach is much more straightforward, elegant and easy to understand. This solution is an approximate one because the independent assumption does not hold true for the Fork-Join queuing network model, in which the task arrival processes at different queuing servers are perfectly synchronized and hence, are not independent of one another. This may result in the load estimation lower than the exact one. Nevertheless, as verified by simulation [53], in the case of the M/M/1 Fork nodes, the approximation is pretty accurate, as it is consistently within a margin of error of 10% in terms of tail latency prediction, which translates to a margin of error of less than 2% in terms of load.

Now, substituting Eq 5.3 into Eq 5.1 and solving for ρ, we finally have,

$$\rho = 1 + r^{-1} \ln (1 - p^{\frac{1}{N}}) \tag{5.4}$$

Since both the job interarrival time and task service time distributions are found to be longer tailed than the exponential one in practice, as mentioned earlier, this solution can serve as a rough upper bound on the achievable load, ρ, at a given tail-latency SLO in terms of $\{r, p\}$ and fanout degree N.



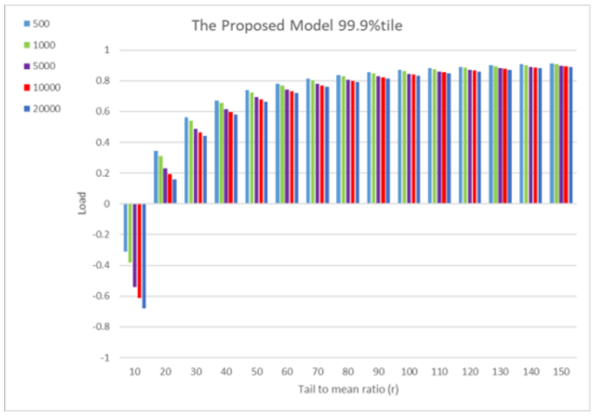


Figure 5.2: Maximum system utilization to achieve the $90th$ and $99.9th$ tail latency with various tail-mean ratio at five different fanout degrees, i.e., 500, 1000, 5000, 10000 and 20000.

5.2 Analysis

In this section, we perform numerical analyses of Eq 5.4. Figure 5.2 depicts the load, ρ, as a function of the tail-to-mean ratio, r, at five different fanout degrees, i.e., $K = 500, 1000, 5000, 10000$ and 20000 for $p = 90$ and 99.9 in the left and right subplots, respectively. These values cover wide parameter ranges of practical interests [75]. We note that when the load that satisfies Eq 5.4 becomes zero or negative, the corresponding tail-latency SLO cannot be met. We include the negative data points in both subplots to help identify how far it is from being able to meet the tail latency SLO. For example, for both $p = 90$ and 99.9 cases

(i.e., the left and the right subplots, respectively), the tail latency SLO cannot be met at $r = 10$ and $K = 5000$, as the corresponding load bars are both negative. However, since the load bar for the $p = 90$ case is much shorter than that of the $p = 99.9$ case, one can expect that the former is much closer to meet the tail-latency SLO than the latter; the tail latency SLO in terms of $\{r, p\} = \{10, 90\}$ is much looser than that of $\{r, p\} = \{10, 99.9\}$.

With the above preparation, we can now explore important features of our model. First, we note that $r = 10$ is too tight to achieve reasonable computing resource utilization in the entire parameter range in terms of both p and N. Both subplots strongly suggest that r needs to be at least around 30 to be effective - assuming the workload takes an exponential behavior. For example, in Google search, the average web indexing at a worker is about $4.2ms$ [30]. This means that striving to achieve a $99th$-percentile tail latency of $150ms$ [65] leads to a r of 36, or a load upper bound of about 60%, regardless cluster size. This means that if the cluster load for an OLDI service already reaches $30 - 50\%$, and assuming no intervene from a dynamically-efficient scheduling algorithm, the room for further load improvement is actually quite limited, given that the tail behavior of production workloads is longer than that of the exponential one.

Second, under the assumption of best case scenario of workload, i.e., exponential behavior, the achievable load ramps up quickly as r increases to 30 but then increases slower afterwards, eventually leveling off as r increases further. Moreover, as r increases, the achievable load saturates at about $75-85\%$ in the range of $p = 90-99.9$ and becomes less sensitive to N as r becomes larger.

In summary, at best case scenarios, to achieve the best trade-offs between the system resource utilization and the tightness of the tail latency SLO, one should set r in the range of $[30-50]$[1] and avoid setting r below 20. However, although its help identify the ceiling bar for workload types, we should be aware that this is not a suitable range for long tail workloads. As it will be explained in the validation section, we learn that a need to implement better tail cutting techniques and/or efficient scheduling algorithms, which could overcome existed high variance in workload, to achieve this best case scenario of N forked tasks.

[1]In the case of the achieved resource utilization in Google cluster that we mentioned before, i.e., the load (ρ) found to be $\leq 25\%$ for CPU and $\leq 50\%$ for memory, and as in Figure 5.2, if *tail-to-mean*-ratio (r) used = 70, that means they are not doing a good job in the matter of managing cluster resources. They should achieve higher utilization for that ratio, e.g., (ρ) should be about or even greater than 80% at *tail-to-mean*-ratio (r) = 70 for p = 99.9 percentile and 90 percentile, respectively.

5.3 Validation Experiments

Workloads types: The existence of high variance in processing a task is terrible when it comes to garnering all elements for better resource utilization. For example, if a specific target-tail response time needs to be maintained, considering the occurrence of high variance in tasks executions as well queues lengths, it become hard to control. Taking the significant difference caused in existing tail response times into account, more overprovisioned resources needed, to maintain the pth responses below or equal to the required target-tail and avoid long waitings in queues. Consequently, the utilization would become worse, as guessed. This condition is contrary to other circumstances when a low variance in system workload is dominating the workload sojourn times. Therefore for the validation purpose, there was a need to experiment a contradicting behavior of two uniquely selected types of workloads. Due to its high variance, Pareto distribution elected to be used as a good model for imitating workload of high variances, which its variance is easy to control as changing the value of alpha parameter. While exponential distribution for its low variance, that could results in better resource utilization, picked to be used in imitating the best-case-scenario of workload. In particular, we consider the following distributions:

– A light-tailed Exponential distribution (Exp) whose CDF is defined as [29]

$$F_{X_i}(x) = 1 - e^{-\frac{x}{\mu}} \quad x \geq 0, \tag{5.5}$$

where μ is the mean service time ($\mu > 0$),

– A heavy-tailed truncated Pareto distribution (TPa) [29] whose CDF is given by

$$F_{X_i}(x) = \frac{1 - (L/x)^{\alpha}}{1 - (L/H)^{\alpha}}, \tag{5.6}$$

where α is a shape parameter; L and H are lower and upper bounds, respectively.

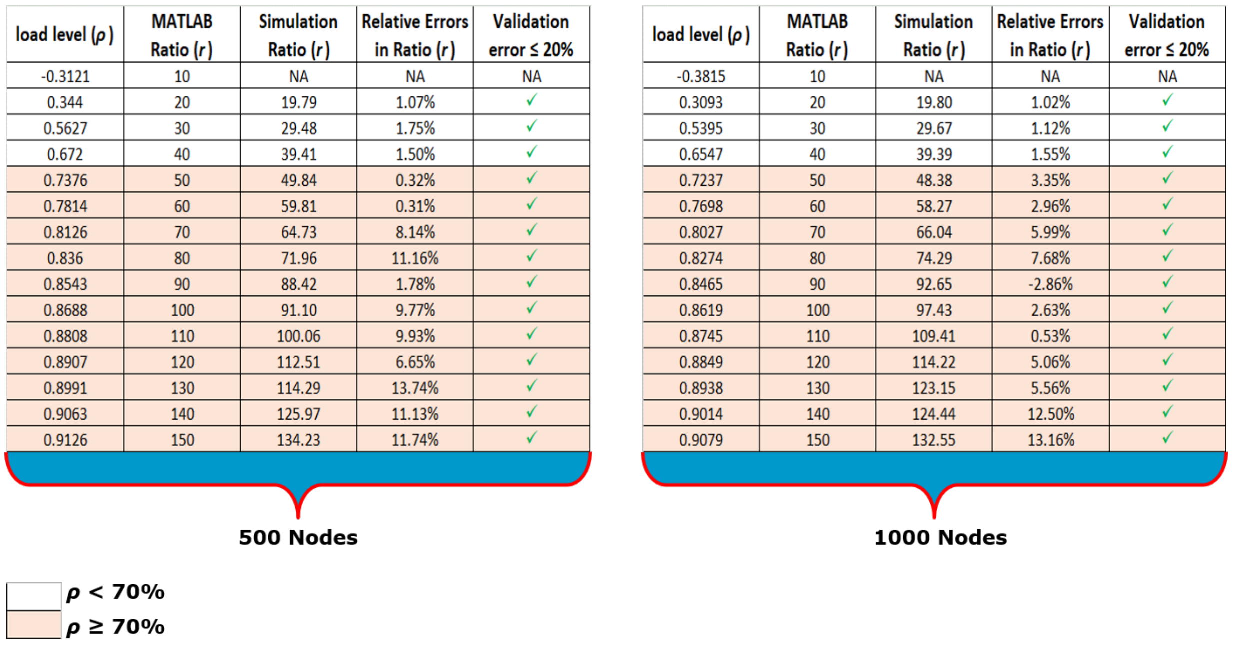
Validation strategy: To validate the correctness of this model, while following the context of our original work, in previous chapters [2, 3, 4], of predicting the tail latency, the observed results using theoretical model can be supported following two typical approaches: numerical analysis and simulation measurements. Using the latter to confirm the results when applying same ratios (r)'s was a little bit subtle for two main reasons. First, in the realm of simulation measurements, where $\rho \geq 0$, ratios of negative load regions cannot be conducted and measured as to the ones encountered in the current derived-model, see figure 5.2. Therefore, the simulated-experiments were forced to be limited just for the *tail-to-mean*-ratios that result in loads ρ's > 0. Second, the implementation-nature of FJQNs-simulations dictates determining load regions as pre-setting to the beginning of any simulation experiment before collecting the final statistical measurements and the required tail-percentiles. Consequently, as to the nature of simulation-measurements that expected not to provide the exact target-ratios, we resorted to applying linear interpolation on the all captured measurements, see tables in Figs 5.3, 5.4, to achieve the exact required r-ratios. Also, it worth asserting that the experimented-simulations end with approximated measurements that are based on a 95% confidence level, which means an applied margin of errors should be expected. Lastly, we limit all the validation experiments to only two different cluster sizes, i.e., N = [500, 1000], due to experienced cost of having long-running times operating FJQNs-simulations.

We report the results for the systems of the two different sizes of Fork nodes, i.e., $N = 500$, and 1000, and running over different loads ρ's based on the determined ratios, assuming that the tasks spawned from all the incoming jobs are randomly dispatched to N nodes in the system, i.e., $k = N$. Specifically, we consider these scenarios:

– **Workload scenario 1**–Exponential distribution with mean service times, $\mu = 4.22ms$.

– **Workload scenario 2**–Truncated Pareto distribution with coefficient of variation $CV = 1.2$ and mean service times, $\mu = 4.22ms$, which results in the distribution parameters $\alpha = 2.0119, L = 2.14ms, H = 276.63ms$.

Following the numerical approach, the same distributions parameters were employed against the simulated ones for the validation purposes. This numerical approach using our



500 Nodes

load level (ρ)	MATLAB Ratio (r)	Simulation Ratio (r)	Relative Errors in Ratio (r)	Validation error ≤ 20%
-0.3121	10	NA	NA	NA
0.344	20	19.79	1.07%	✓
0.5627	30	29.48	1.75%	✓
0.672	40	39.41	1.50%	✓
0.7376	50	49.84	0.32%	✓
0.7814	60	59.81	0.31%	✓
0.8126	70	64.73	8.14%	✓
0.836	80	71.96	11.16%	✓
0.8543	90	88.42	1.78%	✓
0.8688	100	91.10	9.77%	✓
0.8808	110	100.06	9.93%	✓
0.8907	120	112.51	6.65%	✓
0.8991	130	114.29	13.74%	✓
0.9063	140	125.97	11.13%	✓
0.9126	150	134.23	11.74%	✓

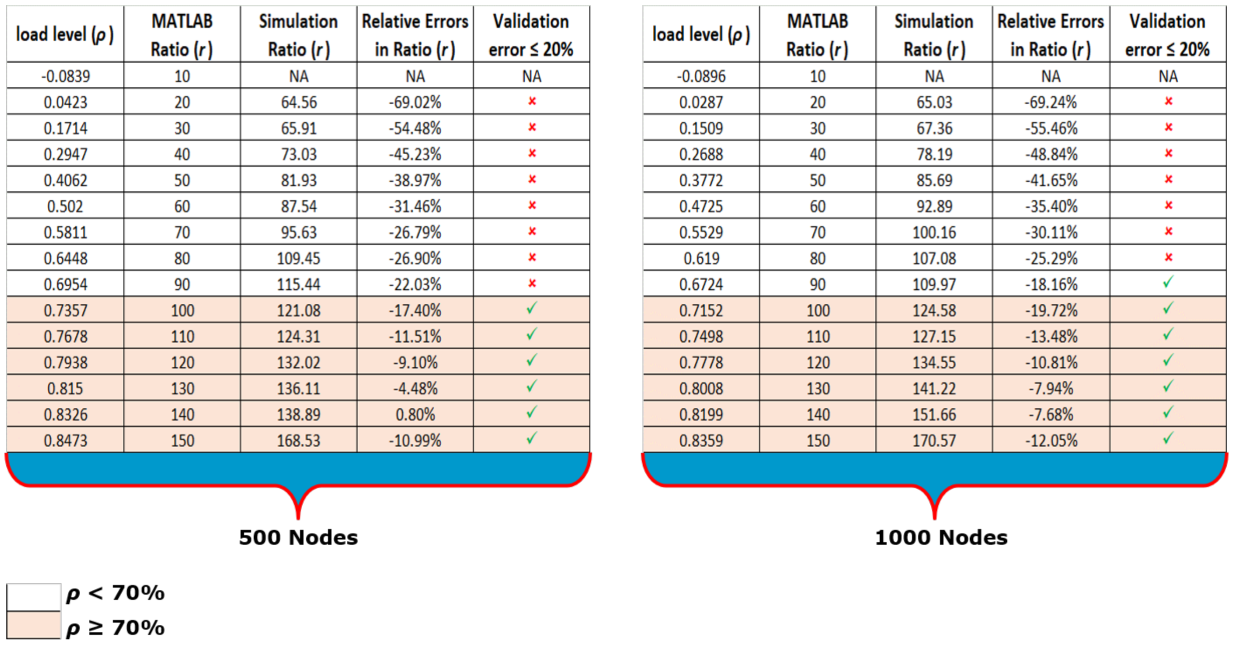
1000 Nodes

load level (ρ)	MATLAB Ratio (r)	Simulation Ratio (r)	Relative Errors in Ratio (r)	Validation error ≤ 20%
-0.3815	10	NA	NA	NA
0.3093	20	19.80	1.02%	✓
0.5395	30	29.67	1.12%	✓
0.6547	40	39.39	1.55%	✓
0.7237	50	48.38	3.35%	✓
0.7698	60	58.27	2.96%	✓
0.8027	70	66.04	5.99%	✓
0.8274	80	74.29	7.68%	✓
0.8465	90	92.65	-2.86%	✓
0.8619	100	97.43	2.63%	✓
0.8745	110	109.41	0.53%	✓
0.8849	120	114.22	5.06%	✓
0.8938	130	123.15	5.56%	✓
0.9014	140	124.44	12.50%	✓
0.9079	150	132.55	13.16%	✓

□ ρ < 70%

▨ ρ ≥ 70%

Figure 5.3: Using *GE* model for predicting load level and Exponential as its workload at 99.9%tile.



500 Nodes

load level (ρ)	MATLAB Ratio (r)	Simulation Ratio (r)	Relative Errors in Ratio (r)	Validation error ≤ 20%
-0.0839	10	NA	NA	NA
0.0423	20	64.56	-69.02%	✗
0.1714	30	65.91	-54.48%	✗
0.2947	40	73.03	-45.23%	✗
0.4062	50	81.93	-38.97%	✗
0.502	60	87.54	-31.46%	✗
0.5811	70	95.63	-26.79%	✗
0.6448	80	109.45	-26.90%	✗
0.6954	90	115.44	-22.03%	✗
0.7357	100	121.08	-17.40%	✓
0.7678	110	124.31	-11.51%	✓
0.7938	120	132.02	-9.10%	✓
0.815	130	136.11	-4.48%	✓
0.8326	140	138.89	0.80%	✓
0.8473	150	168.53	-10.99%	✓

1000 Nodes

load level (ρ)	MATLAB Ratio (r)	Simulation Ratio (r)	Relative Errors in Ratio (r)	Validation error ≤ 20%
-0.0896	10	NA	NA	NA
0.0287	20	65.03	-69.24%	✗
0.1509	30	67.36	-55.46%	✗
0.2688	40	78.19	-48.84%	✗
0.3772	50	85.69	-41.65%	✗
0.4725	60	92.89	-35.40%	✗
0.5529	70	100.16	-30.11%	✗
0.619	80	107.08	-25.29%	✗
0.6724	90	109.97	-18.16%	✓
0.7152	100	124.58	-19.72%	✓
0.7498	110	127.15	-13.48%	✓
0.7778	120	134.55	-10.81%	✓
0.8008	130	141.22	-7.94%	✓
0.8199	140	151.66	-7.68%	✓
0.8359	150	170.57	-12.05%	✓

□ ρ < 70%

▨ ρ ≥ 70%

Figure 5.4: Using *GE* model for predicting load level and Truncated pareto as its workload at 99.9%tile.

previous GE-prediction model for tail latency, chapters [2, 3, 4], was needed, too, for providing a comprehensive study when compared to the utilization-levels predicted by the model. We determined the distribution parameters ahead of the estimation time. Then we follow-step the following algorithm to predict the upper bound of load level. First, for every provided N-cluster size, we locate the distributions over a range of assumed loads (ρ)'s $= [0, 1]$ with changing steps $= 0.0001$ at each loop[2] $-$ you may choose any level of accuracy. Next, while obtaining the distribution, we search for ρ of the required *tail-to-mean*-ratio $(r) = x_p \ / \ \mu$ that satisfied the determined pth percentile. Finally, we repeat the same for all ratios (r)'s $= x_p$'s / μ.

It worth mentioning that the resulted-loads from numerical analysis plus the same distribution parameters, configured as inputs for the simulation experiments, to ensure that the same ratio (r) (or close) can be achieved for our validation purpose. We apply the assumption of homogeneous FJ condition for all the carried on experiments; therefore, only the homogeneous derivation of the tail latency GE-prediction model was applied; Eq 2.13. We list all three results at the same figure to show the difference.

Key observations: So far, we found that the model serves its purpose as an upper bound for predicting the load level, see figures [5.5, 5.6, 5.7]. Here, we aim to provide our view about how the model act as upper bound by pointing out the significant elements of the errors found in prediction. We find that the results stay consistent with our speculation, where exponential can be predicted very well. And at the same time, the model can serve as upper bound for heavy tail distributions, e.g., Pareto - even if we by assumption increased the alpha parameter and the distribution tail. There are two key observations that we need to

[2]For further explanation, on how it is achieved, the mean of task response time and variance of task response time are analytically estimated for a single node to get the distribution's parameters (i.e., shape and scale of the generalized exponential distribution). Then it used for N-nodes with the requested *tail-to-mean*-ratio (r) to find the load (ρ).

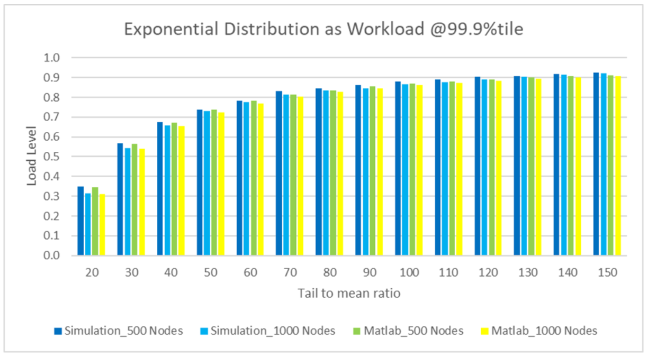


Figure 5.5: Using GE model for predicting load level and Exponential as its workload.

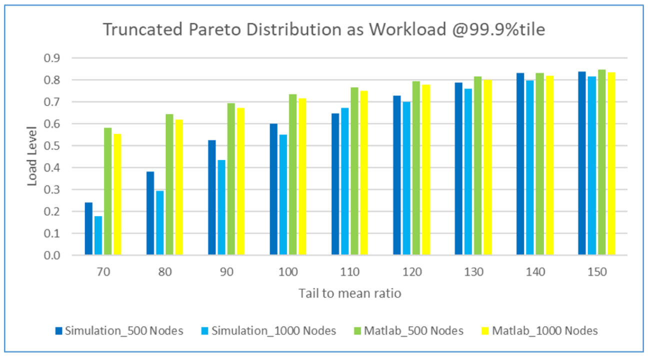


Figure 5.6: Using GE model for predicting load level and Truncated pareto as its workload.

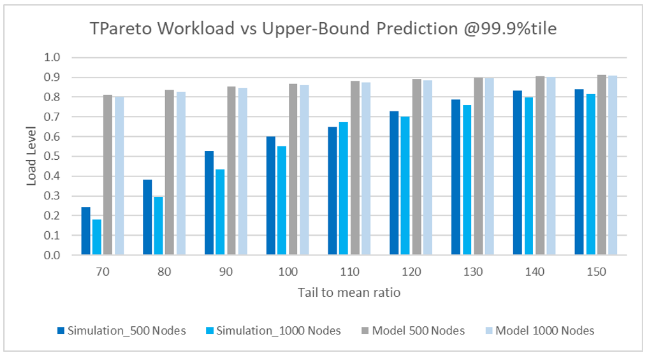


Figure 5.7: load prediction of $GE[Tpareto]$ vs upper bound of proposed model.

acknowledge. First, workload incorporating high variance in its processing when compared to the workload of smaller variance, reveals bad performance in system utilization and tail responses as well. And as we increase the ratio, i.e., relaxing the requested tail SLO, it starts to perform better. Second, when we pay attention to effect of the fanout degree size, for more extended fanout degrees, the load prediction shows worse performance than a smaller fanout degrees. This later observation applies to all studied workload distributions either: light or heavy.

Therefore, the aim is to understand the errors behind both observations. For instance, why heavy Pareto has worse performance than exponential? And why for a longer fanout degree, any workload distribution performs much better than smaller fanout degrees?

First, we have to study the inter-relationship between the elements that play major role in Eq 5.4, see diagram 5.8. For simplicity, let us assume a homogeneous model of the FJ system. For a target tail-to-mean ratio (r_t), if we maintain the same amount of computing resources (R_N) and increase workload variance (Var_w), system utilization (ρ) will decrease due to possible variance in queues lengths. The same holds, i.e., system utilization (ρ) will reduce, if we fixed the workload variance (Var_w), and increased the amount of provisioned resources (R_N), while keeping the origional amount of spawned tasks without change. Further, for the same system condition applied, i.e., all the previous elements: the supplied amount of computing resources (R_N), workload variance (Var_w), and system utilization (ρ) set to fixed values, logically, it is correct to speculate from the reciprocal relationship among the required amount of the $p_{th}\%$ percentile and the needed ratio (r_t) in Eq 5.4. That if we strictly tune the ratio required to a small amount, we have to relax the required percentile of the target tail response time for user satisfaction. This unraveling inter-relationship between the elements of effect in the equation, and on system utilization figure 5.8, demonstrates the level of challenges that are faced by service providers.

Second, we stated that the high variance in task processing ends up with bad resource utilization. Hence, speaking of tail-to-mean ratios and about climbing the ladder of better performance toward achieving high utilization levels. Due to high variance ($CV > 1$), Pareto requires high ratios, r's, to achieve acceptable performance levels in comparison to the expo-

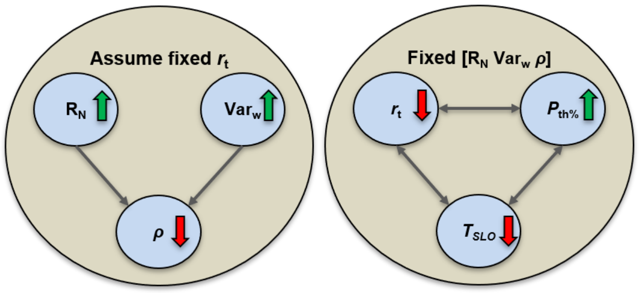


Figure 5.8: Circle of Effects: System prospective VS $Tail_{SLO}$ prospective.

nential, where the exponential for its almost fixed $CV = 1$ gets about the same performance levels for smaller ratios.

Now, to get to understand why, for a more extended fanout degree, load prediction performs worse than smaller fanout degrees, we need to comprehend that this observation makes sense. On the one hand, under the same condition of a load level as we increase the fanouts numbers, the tail also increases in its value. So, looking at Equation 5.4, if the tail for both different fanout degrees kept fixed, mathematically, the effect of larger fanout is going to be reflected negativity at the estimated load level by reciprocal relationship. Hence, the achievable utilization of a more extended fanout degree is going to be lower than a smaller fanout degree. On the other hand, for providing more emphasis on this, practically speaking, it is known that based on the previous observations, there will be high variance among queues lengths at all forked-nodes, i.e., increased chances of having the head of the line blocking phenomena, where at least in one queue node, several long consecutive tasks could mount-up after each one, which cause underutilization for the system as a whole. Once we monitor the entire FJ system and take utilization-snapshots at every moment, it will become apparent the high variance of queues lengths as well as the tasks service-times among nodes-queues, and even though, in the long run, the load (ρ) level maintained fixed for each queuing node. Hence, the result of this momentary underutilization in the FJ system is response times that are dispersed and clustered to form a distribution. Consequently, the tail of this new converged response time distribution becomes longer due to the high variance in jobs-responses, which escalate as we grow the forked-nodes size in par with the forked-tasks amount.

Finally, as we have seen that best case of workload is the one of low variance. However this is not always the case in real production environments as it might become worse. Therefore, possible ways for cutting the tail short and use better scheduling algorithms help circumvent such condition if existed.

5.4 Discussion and Future Work

Improve ceiling bar of Cluster Load: The previous model analysis, of best workload scenario, indicates that to achieve a reasonably high cluster resource utilization, the tail-to-mean ratio, r, must be kept in the range of $[30-50]$. Then the question is: what if a stringent tail-latency SLO must be enforced, despite the fact that r is smaller than 30? Without considering the possibility of service consolidation, perhaps the only effective way to accommodate such stringent tail-latency SLOs is by scaling up workers, or equivalently, reducing mean task service time, S, and hence, increasing r. This, unfortunately, is a brute-force solution.

One possible alternative solution is to cut the tail for task response time shorter by load balancing andor the multiple issuing of a task to a small number of worker replicas [47]. However, this solution may not be very effective. First, we note that the multiple issuing of a task only works when the tail is very long, e.g., due to stragglers [76]. For task response time distributions that are already short tailed, the multiple issuing of a task may no longer be able to further reduce the tail, but rather waste worker resources, as the probability for multiple issues to be in service simultaneously increases as the tail becomes shorter. Then what about load balancing? Before answering this question, we first derive an interesting property for the tasks at each M/M/1 Fork queue.

Denote R_t and T_t as the mean task response time (i.e., the mean task queuing time plus S) and the pth-percentile task tail latency, respectively. The mean response time (using the Little's law):

$$R_t = \frac{1}{\mu(1-\rho)} \tag{5.7}$$

With the task response time distribution given in Eq. 5.4, it can be easily shown that,

$$T_t = \frac{1}{\mu(1-\rho)} \cdot \ln\left(\frac{100}{100-p}\right) \tag{5.8}$$

And finally, we have,

$$\frac{T_t}{R_t} = \ln\left(\frac{100}{100-p}\right) \tag{5.9}$$

Namely, the ratio of the task tail latency and task mean response time only depends on p, independent of load ρ.

Interestingly, at $p = 95$, this ratio is about 3, coinciding with the measured ratio for a leaf server in a google search engine reported in [54]. This leaf server is composed of 16 cores, which form a group of worker replicas. Incoming tasks are first placed in a FIFO queue and the tasks that reach the head of the queue are load balanced among these cores. Hence, the server can be modeled as a G/G/16 queuing server. In general, one can expect that a G/G/16 queue should give a smaller ratio than that of a G/G/1 queue [77], due to resource pooling. However, if no better load balancing approach that take into consideration variance in tasks executions, the fact that the measured ratio is almost the same as that of a M/M/1 queue, strongly suggests that traditional load balancing approach has limited effect on further enhancing the cluster resource utilization.

The Case for $K_i \leq N$: The current solution cannot deal with the case where different jobs may have different fanout degrees, which is fairly common in the case of OLDI social networking services. So an interesting research direction is to develop models to capture the performance bounds for such workloads.

5.5 Related Work

No exact solution to the general FJQN is available. Approximation solutions for job response time distributions are available for FJQNs with M/M/1 [42] and M/M/k [43] Fork nodes. Also, computable stochastic bounds on job waiting and response time distributions for both blocking and non-blocking FJQN are given in [43]. The approximate tail latency solutions for FJQNs based on the analytical results from single-node and two-node cases are also derived [13]. These solutions, however, cannot be expressed in a simple closed form as the one presented in this paper. More recently, black-box approximate tail-latency solutions for a wide range of FJQNs were obtained [53], which led to the same solution given in this paper in the case of the FJQN with M/M/1 Fork nodes. However, the current derivation is much more straightforward, elegant and easy to understand.

To the best of our knowledge, the work presented in this paper is the first one that applies FJQN to the analysis of the job tail-latency bounds for OLDI services. Most existing works on the tail impact on the performance of OLDI services are measurement-based and focused exclusively on a single Fork node, e.g., [78], [30]. Most relevant to the current work is the work on the study of the tail latency impact on the datacenter multicore server speedup [79]. Like our work, it is based on the job response time distribution for a M/M/1 queuing model. However this work is exclusively focused on a single Fork node, as mentioned previously.

5.6 Conclusion

Jobs for online data-intensive (OLDI) services may fan out a large number of tasks to be processed at different workers in a datacenter cluster, while having to meet stringent taillatency Service Level Objectives (SLOs). In this research work, we derive a cluster load bound for a special class of OLDI services. We use a simple Fork-Join queuing network model with M/M/1 Fork nodes to characterize the performance upper bound of such services, assuming that all the jobs have the same fanout degree. We find that due to the queuing effect, the tail latency is, in general, difficult to contain without throwing a significant amount of cluster resources to the problem, which would result in low cluster resource utilization.

REFERENCES

[1] S. Alesawi, M. Nguyen, H. Che, and A. Singhal, “Tail latency prediction for datacenter applications in consolidated environments,” in *2019 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2019, pp. 265–269.

[2] M. Jeon, S. Kim, S. Hwang, Y. He, S. Elnikety, A. L. Cox, and S. Rixner, “Predictive Parallelization: Taming Tail Latencies in Web Search,” in *SIGIR ’14*, 2014, pp. 253–262.

[3] G. Blake and A. G. Saidi, “Where does the time go? Characterizing tail latency in memcached,” in *ISPASS ’15*, 2015, pp. 21–31.

[4] J. Brutlag, “Speed Matters for Google Web Search,” 2009.

[5] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, “Heterogeneity and Dynamicity of Clouds at Scale,” in *SoCC ’12*, 2012, pp. 7:1–7:13.

[6] C. Delimitrou and C. Kozyrakis, “Quasar: Resource-Efficient and QoS-aware Cluster Management,” in *ASPLOS ’14*, 2014, pp. 127–144.

[7] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. Wiley-Interscience, 2006.

[8] A. Thomasian, “Analysis of Fork/Join and Related Queueing Systems,” *ACM Computing Surveys*, vol. 47, no. 2, pp. 1–71, 2014.

[9] R. Nelson and A. N. Tantawi, “Approximate Analysis of Fork/Join Synchronization in Parallel Queues,” *IEEE Transactions on Computers*, vol. 37, no. 6, pp. 739–743, 1988.

[10] E. Varki, “Response time analysis of parallel computer and storage systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 11, pp. 1146–1161, 2001.

[11] S. S. Ko and R. F. Serfozo, "Response Times in M/M/s Fork-Join Networks," *Advances in Applied Probability*, vol. 36, no. 3, pp. 854–871, 2004.

[12] R. J. Chen, "A hybrid solution of fork/join synchronization in parallel queues," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 8, pp. 829–845, 2001.

[13] Z. Qiu, J. F. Pérez, and P. G. Harrison, "Beyond the Mean in Fork-Join Queues: Efficient Approximation for Response-Time Tails," *Performance Evaluation*, vol. 91, pp. 99–116, 2015.

[14] J. Dean and L. A. Barroso, "The Tail at Scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[15] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low Latency via Redundancy," in *CoNEXT '13*, 2013, pp. 283–294.

[16] N. J. Yadwadkar, G. Ananthanarayanan, and R. Katz, "Wrangler: Predictable and Faster Jobs using Fewer Resources," in *SoCC '14*, 2014, pp. 26:1–26:14.

[17] 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, "Scaling Memcache at Facebook," in *NSDI '13*, 2013, pp. 385–398.

[18] P. Delgado, F. Dinu, A. M. Kermarrec, and W. Zwaenepoel, "Hawk: Hybrid Datacenter Scheduling," in *USENIX ATC '15*, 2015, pp. 499–510.

[19] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM computer communication review*, vol. 39, no. 1, pp. 68–73, 2008.

[20] J. F. C. Kingman and M. F. Atiyah, "The single server queue in heavy traffic," *Proceedings of the Cambridge Philosophical Society*, vol. 57, pp. 902–904, 1961.

[21] J. Köllerström, "Heavy Traffic Theory for Queues with Several Servers. I," *Journal of Applied 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 *USENIX HotCloud'16*, 2016.

[23] D. Meisner, C. M. Sadler, A. L. Barroso, W. D. Weber, and T. F. Wenisch, "Power Management of Online Data-Intensive Services," in *ISCA '11*, 2011, pp. 319–330.

[24] S. Sani and O. A. Daman, "Mathematical Modeling in Heavy Traffic Queuing Systems," *American Journal of Operations Research*, vol. 4, pp. 340–350, 2014.

[25] R. D. Gupta and D. Kundu, "Generalized Exponential Distributions," *Australian & New Zealand Journal of Statistics*, vol. 41, no. 2, pp. 173–188, 1999.

[26] L. Kleinrock, *Queueing Systems, Vol. 1: Theory.* Wiley-Interscience, 1975.

[27] "http://crystal.uta.edu/ hche/publications/forktail-ext.pdf," Extended version.

[28] 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.

[29] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*, 1st ed. Cambridge University Press, 2013.

[30] D. Meisner, W. Junjie, and T. F. Wenisch, "BigHouse: A Simulation Infrastructure for Data Center Systems," in *ISPASS '12*, 2012, pp. 35–45.

[31] "https://spark.apache.org," Apache Spark.

[32] "https://hadoop.apache.org," Apache Hadoop.

[33] Y. Chen, S. Alspaugh, and R. Katz, "Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1802–1813, Aug 2012.

[34] 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 *EuroSys '10*, 2010, pp. 265–278.

[35] L. Flatto and S. Hahn, "Two Parallel Queues Created by Arrivals with Two Demands I," *SIAM Journal on Applied Mathematics*, vol. 44, no. 5, pp. 1041–1053, 1984.

[36] F. Alomari and D. A. Menasce, “Efficient Response Time Approximations for Multiclass Fork and Join Queues in Open and Closed Queuing Networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1437–1446, 2014.

[37] S. Balsamo, L. Donatiello, and N. M. Van Dijk, “Bound performance models of heterogeneous parallel processing systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 10, pp. 1041–1056, 1998.

[38] R. J. Chen, “An Upper Bound Solution for Homogeneous Fork/Join Queuing Systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 5, pp. 874–878, 2011.

[39] D. Wang, G. Joshi, and G. Wornell, “Efficient Task Replication for Fast Response Times in Parallel Computation,” *arXiv:1404.1328*, pp. 1–20, 2014.

[40] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytiä, and A. Scheller-Wolf, “Reducing Latency via Redundant Requests: Exact Analysis,” in *SIGMETRICS ’15*, 2015, pp. 347–360.

[41] Z. Qiu and J. F. Perez, “Evaluating the Effectiveness of Replication for Tail-Tolerance,” in *CCGrid ’15*, 2015, pp. 443–452.

[42] S. Balsamo and I. Mura, “Approximate response time distribution in Fork and Join systems,” in *SIGMETRICS ’95/PERFORMANCE ’95*, 1995, pp. 305–306.

[43] A. Rizk, F. Poloczek, and F. Ciucu, “Computable Bounds in Fork-Join Queueing Systems,” in *SIGMETRICS ’15*, 2015, pp. 335–346.

[44] T. Zhu, A. Tumanov, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger, “PriorityMeister: Tail Latency QoS for Shared Networked Storage,” in *SoCC ’14*, 2014, pp. 29:1–29:14.

[45] A. Wang, S. Venkataraman, S. Alspaugh, R. Katz, and I. Stoica, “Cake: Enabling High-level SLOs on Shared Storage Systems,” in *SoCC ’12*, 2012, pp. 14:1–14:14.

[46] A. D. Ferguson, P. Bodik, E. Boutin, and R. Fonseca, "Jockey: Guaranteed Job Latency in Data Parallel Clusters," in *EuroSys '12*, 2012, pp. 99–112.

[47] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[48] W. Chen, J. Rao, and X. Zhou, "Preemptive, low latency datacenter scheduling via lightweight virtualization," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 251–263.

[49] V. Jalaparti, P. Bodik, S. Kandula, I. Menache, M. Rybalkin, and C. Yan, "Speeding up distributed request-response workflows," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 219–230.

[50] K. Karanasos, S. Rao, C. Curino, C. Douglas, K. Chaliparambil, G. M. Fumarola, S. Heddaya, R. Ramakrishnan, and S. Sakalanaga, "Mercury: Hybrid centralized and distributed scheduling in large shared clusters." in *USENIX Annual Technical Conference*, 2015, pp. 485–497.

[51] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: distributed, low latency scheduling," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 69–84.

[52] M. Nguyen, Z. Li, F. Duan, H. Che, and H. Jiang, "The tail at scale: how to predict it?" in *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*. USENIX Association, 2016.

[53] 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 *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2018, pp. 206–217.

[54] D. Meisner, C. M. Sadler, A. L. Barroso, W. D. Weber, and T. F. Wenisch, "Power Management of Online Data-Intensive Services," in *Proceedings of the 38th annual In-*

ternational Symposium on Computer Architecture, ser. ISCA '11. ACM, 2011, pp. 319–330.

[55] J. Kingman, "The single server queue in heavy traffic," in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 57, no. 04. Cambridge Univ Press, 1961, pp. 902–904.

[56] J. Köllerström, "Heavy traffic theory for queues with several servers. i," *Journal of Applied Probability*, vol. 11, no. 03, pp. 544–552, 1974.

[57] R. D. Gupta and D. Kundu, "Theory & methods: Generalized exponential distributions," *Australian & New Zealand Journal of Statistics*, vol. 41, no. 2, pp. 173–188, 1999.

[58] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou, "Apollo: Scalable and coordinated scheduling for cloud-scale computing." in *OSDI*, vol. 14, 2014, pp. 285–300.

[59] P. Delgado, F. Dinu, A.-M. Kermarrec, and W. Zwaenepoel, "Hawk: Hybrid datacenter scheduling," in *Proceedings of the 2015 USENIX Annual Technical Conference*, no. CONF. USENIX Association, 2015.

[60] P. Delgado, D. Didona, F. Dinu, and W. Zwaenepoel, "Job-aware scheduling in eagle: Divide and stick to your probes," in *Proceedings of the Seventh ACM Symposium on Cloud Computing.* ACM, 2016, pp. 497–509.

[61] Z. Liu and S. Cho, "Characterizing machines and workloads on a google cluster," in *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on.* IEEE, 2012, pp. 397–403.

[62] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca, "Jockey: guaranteed job latency in data parallel clusters," in *Proceedings of the 7th ACM european conference on Computer Systems.* ACM, 2012, pp. 99–112.

[63] A. Wang, S. Venkataraman, S. Alspaugh, R. Katz, and I. Stoica, "Cake: enabling high-level slos on shared storage systems," in *Proceedings of the Third ACM Symposium on Cloud Computing*. ACM, 2012, p. 14.

[64] T. Zhu, A. Tumanov, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger, "Prioritymeister: Tail latency qos for shared networked storage," in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2014, pp. 1–14.

[65] T. Zhu, M. A. Kozuch, and M. Harchol-Balter, "Workloadcompactor: Reducing datacenter cost while providing tail latency slo guarantees," in *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 2017, pp. 598–610.

[66] S. Alesawi and S. Ghanem, "Overcome heterogeneity impact in modeled fork-join queuing networks for tail prediction," in *2019 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2019, pp. 270–275.

[67] L. A. Barroso, J. Dean, and U. Holzle, "Web search for a planet: The google cluster architecture," *IEEE micro*, vol. 23, no. 2, pp. 22–28, 2003.

[68] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3, pp. 158–169, 2016.

[69] D. Meisner, J. Wu, and T. F. Wenisch, "Bighouse: A simulation infrastructure for data center systems," in *Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 35–45.

[70] N. S. Rao, "Vector space methods for sensor fusion problems," *OPTICAL ENGINEERING-BELLINGHAM-INTERNATIONAL SOCIETY FOR OPTICAL ENGINEERING-*, vol. 37, pp. 499–504, 1998.

[71] Q. Wu, N. S. Rao, and S. S. Iyengar, "On transport daemons for small collaborative applications over wide-area networks," in *PCCC 2005. 24th IEEE International Performance, Computing, and Communications Conference, 2005*. IEEE, 2005, pp. 159–166.

[72] A. Thomasian and A. N. Tantawi, "Approximate Solutions for M/G/1 Fork/Join Synchronization," in *Proceedings of the 26th Conference on Winter Simulation*, ser. WSC '94, 1994, pp. 361–368.

[73] "M/M/1 Queue," https://en.wikipedia.org/wiki/M/M/1_queue.

[74] "Extreme Value Theorem," https://en.wikipedia.org/wiki/Extreme_value_theorem.

[75] S. Shen, V. v. Beek, and A. Iosup, "Statistical characterization of business-critical workloads hosted in cloud datacenters," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2015, pp. 465–474.

[76] D. Wang, G. Joshi, and G. Wornell, "Using straggler replication to reduce latency in large-scale parallel computing," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 3, pp. 7–11, 2015.

[77] "G/G/1 Queue," https://en.wikipedia.org/wiki/G/G/1_queue.

[78] D. Meisner, C. M. Sadler, A. L. Barroso, W. D. Weber, and T. F. Wenisch, "Power Management of Online Data-Intensive Services," in *Proceedings of the 38th annual International Symposium on Computer Architecture*, ser. ISCA '11. ACM, 2011, pp. 319–330.

[79] C. Delimitrou and C. Kozyrakis, "Amdahl's law for tail latency," *Communications of the ACM*, vol. 61, no. 8, pp. 65–72, 2018.

BIOGRAPHICAL STATEMENT

Sami Alesawi was born in Madinah, Saudi Arabia, in late 70s. He received his bachelor's degree in Computer Engineering from the Faculty of Engineering at King Abdulaziz University (KAU), fall 2000. He gained his master's degree in computer science from the Faculty of Computing & Information Technology - KAU Spring 2009. He received a full scholarship from his university and continued his doctoral studies in the USA, and received his Ph.D. degrees from The University of Texas at Arlington in 2020. Since 2002 he was with the Department of Electrical Engineering, KAU University working as a Lab Instructor and IT unit manager. Before that, he worked as a network engineer in the private sector, and after he completed his master's degree, he worked as a lecturer at King Abdulaziz University in Rabigh. His current research interests in datacenter resource management and job scheduling.