PERFORMANCE ANALYSIS OF SCALE-OUT WORKLOADS ON PARALLEL AND DISTRIBUTED SYSTEMS

by

MINH QUANG NGUYEN

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy at
The University of Texas at Arlington
Doctor of Philosophy
August, 2017

Arlington, Texas

ABSTRACT

PERFORMANCE ANALYSIS OF SCALE-OUT WORKLOADS

ON PARALLEL AND DISTRIBUTED SYSTEMS

Minh Quang Nguyen, Ph.D.

The University of Texas at Arlington, 2017

Supervising Professor: Hao Che

Scale-out applications have emerged to be the predominant datacenter workloads. The request processing workflow for such a workload may consist of one or more stages with massive numbers of compute nodes for parallel data-intensive processing. As a classic model for the most essential building block of a workflow, the Fork-Join queuing network model is found to be notoriously hard to solve due to the involvement of task partitioning and merging with barrier synchronization. The work in this dissertation aims to develop approximation methods for the prediction of tail and mean latency for Fork-Join queuing networks in a high load region, where resource provisioning is desirable.

Specifically, this dissertation makes the following contributions. First, we propose a simple prediction model for tail latency for a large class of Fork-Join queuing networks of practical importance in a high load region using only the mean and variance of response times for tasks in the Fork phase as input. We also generalize the model for the cases where each request processing includes only a partial number of processing units in the network. The prediction errors for the 99th percentile request latency are found to be consistently within 10% at the load of 90% for both model and measurement-based testing cases. This work thus establishes a link between the system-level request tail latency constraint, a.k.a. the tail-latency Service Level Objective (SLO), and the subsystem-level task performance requirements, facilitating explicitly tail-constrained resource provisioning at scale. Second, we propose an empirical approach for mean latency approximation for such systems based on the developed tail latency prediction model. The experimental results show that the approach gives accurate predictions for various testing cases when the system is at high loads. Finally, we put forward a framework for scaling analysis of scale-out applications. With the proposed solution for mean latency approximation, we are able to derive a closed-form solution for this framework that can be used for the exploration of scaling properties. A case study is given for the case where all the task service time distributions are exponential.

ACKNOWLEDGMENTS

I would like to thank my supervising professor, Dr. Hao Che, for motivating and encouraging me and also for his invaluable advice during the course of my doctoral studies. It has been a great pleasure to work with him on research projects. I have learned a lot through useful discussions with him during my research.

I would also like to express my gratitude to Dr. Ramez Elmasri, Dr. Jeff (Yu) Lei, and Dr. Hong Jiang for their interest in my research and for taking time to serve on my dissertation committee.

I owe a debt of gratitude to all of my friends for their help and advice during my studies at UTA. In particular, I would like to thank Nha Nguyen for helping me when I first started my doctoral program and sharing with me long discussions on various topics. I am grateful to Zhongwei Li and Feng Duan for their support with experiments on cloud services. I wish to thank Dongchul Kim, Shuo Li, Mingon Kang, Ashis Biswas for their friendship and helpful discussions.

Finally, I would like to give special thanks to my wife, Phuong Pham, for all the support and patience during my journey to the doctoral degree.

To my mother and my mother-in-law, who supported us during my studies

To my wife, Phuong Pham, and my lovely kids, Khoi and Clara, who inspired me to complete this dissertation

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

BCMP	A theorem developed by Baskett, Chandy, Muntz, and Palacios
CDF	Cumulative Distribution Function
CV	Coefficient of Variation
DAG	Directed Acyclic Graph
FCFS	First Come First Served
FJQN	Fork-Join Queuing Network
IS	Infinite Server
LCFS-PR	Last-Come-First-Served - Preemptive Resume
OLDI	Online Data-Intensive
PDF	Probability Density Function
PS	Processor Sharing
QoS	Quality of Service
RR	Round Robin
SLO	Service Level Objective
VM	Virtual Machine

Chapter 1

INTRODUCTION

Scale-out applications have emerged to be the predominant datacenter workloads. They can be broadly classified into two categories: latency-sensitive applications, a.k.a. online data-intensive (OLDI) applications, e.g., web search, social networking, or machine translation, and throughput-oriented background batch applications, e.g., web indexing or offline image processing [42]. Background batch applications usually have no strict quality-of-service (QoS) constraints [42]. Latency-sensitive applications, on the contrary, provide user-facing services that often require strict service level objectives (SLOs), e.g., responsiveness in sub-second time scale with a minimum throughput guarantee [40,44]. Failure to meet SLOs even with a small margin for user-facing workloads can result in significant customer churn rate and hence revenue loss [14]. Due to the lack of a good understanding of how to translate request SLOs into task budgets, or equivalently resources for task processing, a common practice is to overprovision resources and isolate services to meet tight SLOs, at the cost of low resource utilization, e.g., less than 50% CPU and memory utilizations [12,22,50]. Hence a link between request SLOs and task budgets is needed to facilitate resource provisioning for OLDI workloads.

Request processing in user-facing workloads may consist of one or more stages and generally involves a massive numbers of compute nodes for parallel data-intensive processing. A critical building block in the request processing workflow is the Fork-Join structure, which can be modeled as a Fork-Join queuing network [51]. The Fork-Join queuing network model is found to be notoriously difficult to solve due to the involvement of task partitioning and merging with barrier synchronization and queuing [51,55]. In this dissertation, we aim to develop approximation models for the prediction of tail and mean latency for Fork-Join queuing networks in a high load region, where resource provisioning is desirable. In particular, the aim is to establish a link between request SLOs and task performance budgets, which can be used to facilitate highly scalable, distributed resource provisioning for scale-out applications.

In what follows, we highlight the contributions of the work in this dissertation:

First, we propose a simple model for the prediction of tail latency of Fork-Join queuing networks in a high load region. The model requires only mean and variance of response times on a task processing node, which are used to approximate the response time distribution using the generalized exponential distribution [27]. The request tail latency then can be derived in a closed-form solution. The results show that the model gives accurate predictions for a wide range of experimental settings, which yields errors within 10% for the 99th percentile prediction at the load of 90%. We also extend the model for a more general case when each request is forked to only a partial number of nodes in the system. The extended model also gives accurate predictions for different distributions of the numbers of tasks per request, including fixed number, discrete uniform and triangular distributions.

Second, we propose an empirical approach to estimate the mean response time for Fork-Join queuing network based on the predicted tail latency and the tail weight of service time distribution. The experimental results show that the approach gives accurate predictions for various testing cases when the system is at high loads.

The above approximations for the tail and mean latency set the foundation for development of SLO-guaranteed job scheduling and resource provisioning algorithms for both user-facing interactive and background batch applications.

Finally, we put forward a framework for scaling analysis of scale-out applications. With the proposed solution for mean latency approximation, we are able to derive a closed-form solution for this framework. Preliminary analysis of the solution reveals the richness of the scaling properties of scale-out applications.

The remainder of this dissertation is organized as follows. Chapter 2 proposes the prediction models and analyzes the performance of tail latency prediction. Chapter 3 presents an alternative analytical model for tail latency prediction and an empirical approach for mean latency prediction. Chapter 4 introduces a framework for scaling analysis of scale-out applications. Finally, chapter 5 makes concluding remarks and discusses our future research plan.

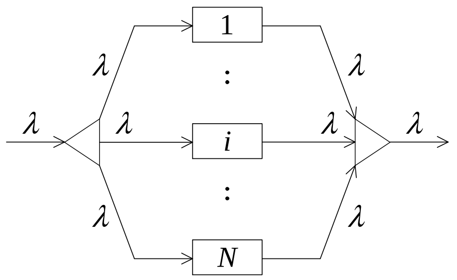
Chapter 2

PREDICTION MODEL FOR TAIL LATENCY AT SCALE

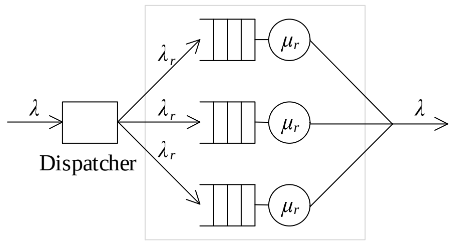
2.1 Introduction

Scale-out, online-data-intensive (OLDI) workloads, such as web searching and social networking, provide user-facing services that involve a large number of servers for parallel processing, while requiring sub-second request responsiveness under high incoming request rates. The system running these workloads usually operates under stringent SLOs, such as imposing a tight tail constraint on high percentile request response time, e.g., 99th or 99.9th-percentile, to satisfy as many user requests as possible [20, 40].

However, imposing tight tail SLO for OLDI workloads makes resource provisioning in datacenter a challenging task. Due to the lack of good understanding of request tail behaviors, the current practice is to overprovision datacenter resources to meet SLO, at the cost of low resource utilization, e.g., less than 50% CPU and memory utilizations [12, 22, 50]. Although resource provisioning proposals with tail SLOs in mind exist, they generally do not incorporate tail SLOs explicitly as design constraints and rely on empirical data to verify whether the design meets tail SLOs or not. For example, the resource provisioning problem is formulated as the minimization of the variance of data flow path latency, as a way to indirectly curtail the tail latency [32]; the target tail latency SLOs are tracked using online dynamic feedback-loop-control-based schedulers [24, 62]; and employing job priority and a rate limiting technique based on the network calculus theory [65]. The root cause of the status quo is due to the lack of a link between system-level request tail SLOs and the subsystem-level task performance requirements. The key difficulty lies in the fact that a scale-out workload may involve *task partitioning* and *merging* as well as *task queuing*. Each request in the request flow with average request rate λ involves tasks to be queued at and processed by up to several thousands of task subsystems in parallel and then all the task results are merged and returned, as depicted in Fig. 2.1a. Here a task subsystem may involve multiple replicated servers for task-level fault tolerance and load balancing, e.g., Fig. 2.1b, where $\lambda_r = \lambda/3$ in the case of load balancing. Notable examples are Web search engines [11] and social networking [47]. In this case, the request response time is determined by the slowest task [20]. As the system scales out, the probability that the request response time may hit the tail task latency quickly increases [20]. To date, no general results are available that can predict the tail request response time at scale. This lack of understanding of request tail



(a) A system with subsystems as black boxes.



(b) A subsystem with one dispatcher and three replicated servers.

Figure 2.1: The task partitioning and merging model.

behaviors is further exacerbated by the various task scheduling and tail-cutting techniques being used in task processing. In particular, as an effective tail-cutting technique, replicated servers in each subsystem are being used to allow redundant task issues to more than one replicated server to be processed, with the earliest result returned and rest removed [20, 61]. Although some analytic results are available on redundant task issues [26, 48, 63], they either address only a single replicated server subsystem with exponential task service time distribution only [26] or parallel request load balancing without task partitioning [48, 63]. The task partitioning-and-merging part of a scale-out workload generally lies in the critical path for request processing and 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 [33]. Hence, it is of paramount importance to establish a link between the system-level request tail SLOs and the subsystem-level task performance requirements to facilitate explicitly tail-constrained resource provisioning at scale.

This chapter aims at tackling the above challenge. This is an extension of the work presented in [46]. It makes the following major contributions. First, by treating each subsystem as a black box, we find that the tail behavior of a task mapped to a subsystem can be captured by a generalized exponential distribution function in the high load region, which uses the mean and variance of the task response time as input. This black-box solution allows the request distribution function and thus any given request tail SLOs to be explicitly expressed as a function of the means and variances of the individual task response times as the system scales out. Hence, in the case of homogeneous subsystems for parallel task processing, the request tail SLO is only dependent on the mean and variance of the task response time for tasks mapped to any given subsystem. Second, we generalize the model for the cases where each request processing includes only a partial number of processing units in the system.

Finally, we discuss how the proposed request tail prediction model may be used to facilitate highly scalable, explicitly tail-constrained resource provisioning using homogeneous virtual machines (VMs) in a cloud environment.

The remainder of the chapter is organized as follows. Section 2.2 presents the prediction model for tail latency. Section 2.3 shows experimental results for both simulated and real systems. Section 2.4 introduces a general model for tail latency prediction. Section 2.5 discusses how the proposed model may facilitate tail-constrained resource provisioning. Finally, Section 2.6 concludes the chapter.

2.2 Tail Latency Prediction Model

A system serving scale-out, OLDI workloads commonly involves a large number of task subsystems for parallel processing. The diversity in the actual implementation of subsystems makes it extremely difficult to predict the task performance, let alone the request performance, in general. However, since the ultimate goal of this research is to be able to design request scheduling algorithms that can meet stringent tail SLOs by proof of design, while achieving high resource utilization, we are interested in the peak-load resource provisioning in a high load region, e.g., 90% or higher. In this region, it is possible to predict the task performance for a task mapped to a wide range of subsystems using a simple prediction model, as we now explain. There is a large body of research results in the context of queuing

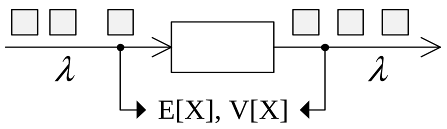


Figure 2.2: A subsystem as a black box.

performance in high load regions (e.g., see [52] and the references therein). In particular, a classic result, known as the central limit theorem for heavy traffic queuing systems [34, 37], states that for a G/G/m (here m is the number of servers) queue under heavy traffic load, the waiting time distribution could 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 waiting time or 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, in this paper, we treat any task subsystem, e.g., the one in Fig. 2.1b, as a black box, given in Fig. 2.2. We further postulate that for a task mapped to a black box subsystem and in the high load region, the task response time distribution $F_X(x)$ for any arrival process can be approximated as a generalized exponential distribution function [27], as follows,

$$F_X(x) = \begin{cases} (1 - e^{-x/\beta})^\alpha & x > 0, \\ 0 & \text{otherwise}, \end{cases} \tag{2.1}$$

where α and β are the shape and scale parameters, respectively. The mean and variance of the task response time are given by [27]

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

$$\mathbb{V}[X] = \beta^2[\psi'(1) - \psi'(\alpha + 1)], \tag{2.3}$$

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

From Eqs. (2.2) and (2.3), it is clear that the distribution in Eq. (2.1) is completely determined by the mean and variance of the task response time. The rationale behind the use of this distribution, instead of the exponential distribution, is that it can capture both heavy-tailed and short-tailed task behaviors depending on the parameter settings and meanwhile, it degenerates to the exponential distribution at $\alpha = 1$ and $\mathbb{E}[X] = \beta$. As we shall see in the following subsection, this distribution significantly outperforms the exponential distribution in terms of tail latency predictive power for all the cases studied.

The implication of the above black box approximation is significant. It allows not only the task performance of a task mapped to a diverse range of subsystems to be captured by a unified distribution function, but also the request response time distribution and hence the tail SLO for the entire task-partitioning-merging system to be derived. To see why this is the case, one notes that with all the task subsystems in Fig. 2.1a being viewed as black boxes, one effectively transforms the task-partitioning-merging problem into a split-and-merge model [38] whose distribution function can be expressed as follows, assuming the task response times for tasks mapped to different subsystems are independent random variables,

$$F_{(N)}(x) = \begin{cases} \prod_{i=1}^{N} (1 - e^{-x/\beta_i})^{\alpha_i} & x > 0, \\ 0 & \text{otherwise,} \end{cases} \tag{2.4}$$

Now assume that the parallel subsystems are homogeneous, the distribution function can be further simplified as,

$$F_{(N)}(x) = \begin{cases} (1 - e^{-x/\beta})^{N\alpha} & x > 0, \\ 0 & \text{otherwise,} \end{cases} \tag{2.5}$$

With Eq. (2.5), it can be easily shown that the pth percentile request response time x_p can be written as,

$$x_p = -\beta \log \left(1 - \left(\frac{p}{100}\right)^{\frac{1}{N\alpha}}\right) \tag{2.6}$$

Since x_p is a function of α and β, which in turn, are functions of $\mathbb{E}[X]$ and $\mathbb{V}[X]$ of the task response time (according to Eqs. (2.2) and (2.3)), a link between any given tail SLO in terms of x_p and p, and $\mathbb{E}[X]$ and $\mathbb{V}[X]$ is established. The implication of this result is significant. On one hand, with any given tail SLO, the resulting $\mathbb{E}[X]$ and $\mathbb{V}[X]$ can serve as the task response time budgets for highly scalable, distributed task-level resource provisioning. On the other hand, with given measured task response time statistics in terms of $\mathbb{E}[X]$ and $\mathbb{V}[X]$, whether the system meets the target tail SLO or not can be accurately predicted. In the following section, we test the performance of this prediction model at the subsystem and system levels, separately.

2.3 Experimental Results

In what follows, we assess the accuracy of our proposed prediction model on different experiment settings, including a single simulated subsystem, a simulated system with and without replication, and a real cloud testbed. The accuracy is given in the form of relative error between the predicted result, t_{pred}, and the simulated one, t_{sim}, i.e.,

$$error = \frac{100(t_{pred} - t_{sim})}{t_{sim}}\,.$$

2.3.1 Scenario 1: Subsystems

In this section, we test the accuracy of the proposed prediction model against a wide range of subsystems including pure model-based and hybrid measurement-and-model-based subsystems. We also compare the accuracy of the model using the generalized exponential distribution against that of the exponential distribution.

Consider a typical subsystem setup given in Fig. 2.1b. It includes a dispatcher and three replicated servers. A task arriving at the subsystem is distributed to server replicas by a dispatcher based on a predetermined policy. Each server replica can be viewed as an M/G/1 queuing system. Namely, for all the cases studied, the task arrival process is modeled as a Poisson process, which is considered a good model for scale-out workloads [44]. Both model-based and measurement-based service time distribution functions are considered, including the following,

– Empirical distribution measured from a Google search test leaf node provided in [43], 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;

– A heavy-tailed truncated Pareto distribution [5] with the same mean service time, i.e., 4.22ms, and a CV of 1.2, resulting in the corresponding parameters: the shape $\alpha = 2.0119$, the lower bound $L = 2.14$ms, and the upper bound $H = 276.6$ms, which is set at the same maximum value of the empirical distribution above.

– Weibull distribution [13] also with the same mean service time and a CV of 1.5, resulting in the corresponding parameters: the shape parameter $\alpha = 0.6848 < 1$, i.e., a heavy-tailed distribution [13], and the scale parameter $\beta = 3.2630$.

We consider two task dispatching policies. The first policy is a simple and popular one, known as the Round-Robin (RR) policy. In this policy, 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 issue, a well-known tail-cutting technique [20, 61]. This policy allows one or more replications of a task to be sent to different server replicas in the subsystem. The replications may be sent in predetermined intervals to avoid overloading the server replicas. In our experiments, at most one task replication can be issued, provided that the original one does not finish within 10ms, which is around the 95th percentile of the empirical distribution above.

We compare the simulated tail task response time against the one predicted by the proposed prediction model, i.e., Eq. (2.1), which uses the simulated mean and variance of the task response time as input. Fig. 2.3 presents the prediction errors for both the exponential and generalized exponential distributions at the load of 90%. First, we note that the generalized exponential distribution significantly outperforms the exponential distribution for all the cases studied. Second, the prediction errors for the generalized exponential distribution are consistently within 10% across the entire 95-99.9th percentile response time range, even for the RR case without tail cutting. These results confirm our postulation that the generalized exponential distribution function could accurately predict the task tail performance in the high load region.

2.3.2 Scenario 2: Simulated Systems

In this section, we evaluate the accuracy of our prediction model as the system scales out. We consider the task-partitioning-merging system in Fig. 2.1a with $N = 10, 100, 500,$ and 1000 subsystems for all the previously studied model-based and hybrid subsystems. We also consider the case when each subsystem has only one processing node, i.e., without replication

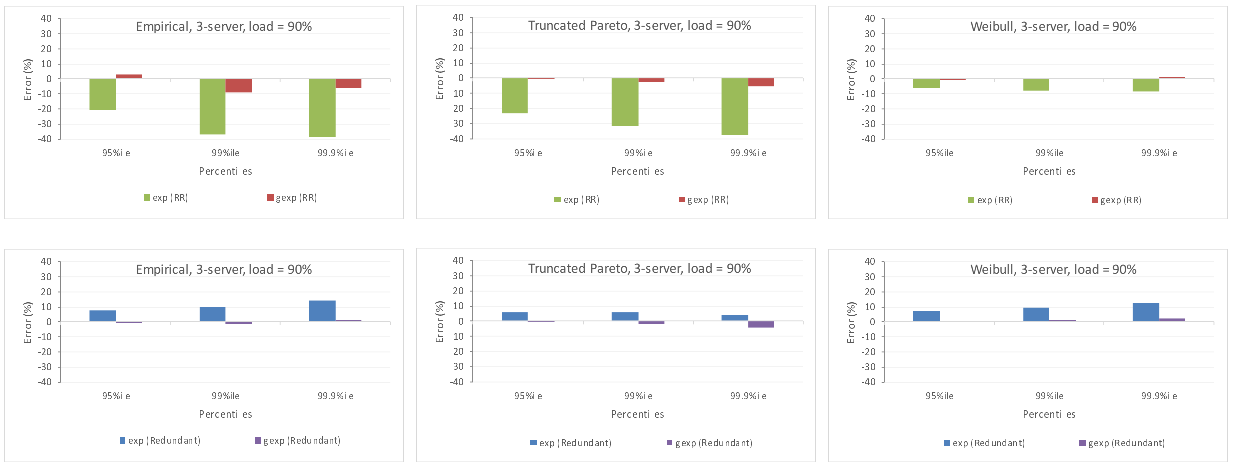


Figure 2.3: Prediction errors for both model-based and hybrid subsystems with Round-Robin (upper three plots) and redundant-task-issue (lower three plots) policies.

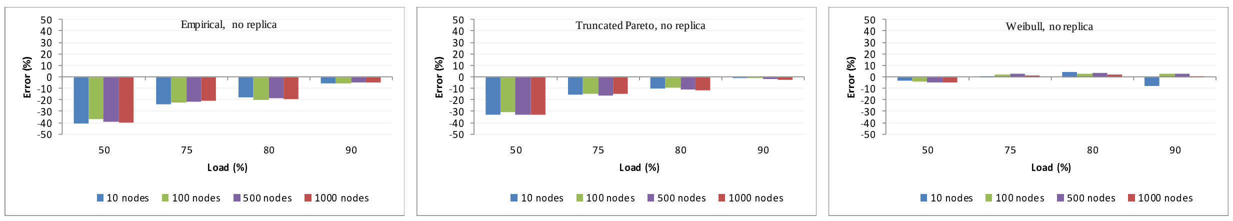


Figure 2.4: Prediction errors for systems composed of both model-based and hybrid non-replicated subsystems.

and dispatching policy. Fig. 2.4 and 2.5 present the prediction errors at different load levels for the 99th percentile request response times for a system with non-replicated subsystems and 3-replica subsystems, respectively. Again, for all the cases studied, the errors are within 10% at the load of 90%. Even at the load of 80%, the prediction errors are with 10% and 20% for the cases with and without tail cutting, respectively.

2.3.3 Scenario 3: Real Scale-out Systems in Cloud

In this section, we assess the accuracy of our proposed prediction model on a real scale-out system with Amazon EC2 instances. We implemented a simple grep-like program on

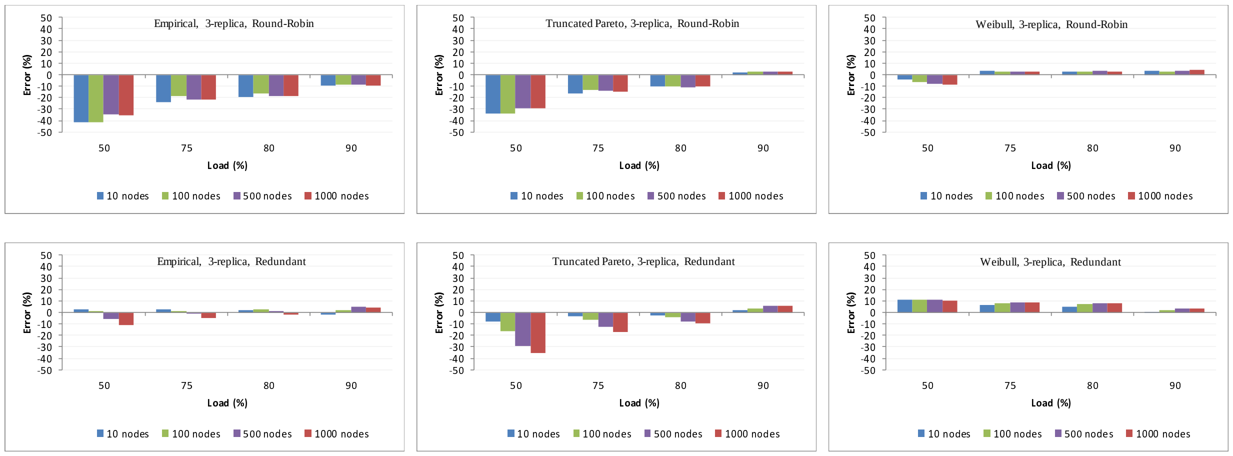


Figure 2.5: Prediction errors for systems composed of both model-based and hybrid replicated subsystems with Round-Robin (upper three plots) and redundant-task-issue (lower three plots) policies.

Apache Spark framework (version 2.1.0) [2] that looks up a keyword in a set of documents and returns a total number of lines containing that keyword. The testing cluster includes one master node using an EC2 c4.4xlarge instance and 32 or 64 worker nodes using EC2 c4.large instances. We used a subset of the English version of Wikipedia as the document for lookup. Each worker node holds a shard of the document whose size is 128MB, corresponding to the default block size on HDFS (Hadoop Distributed File System) [1]. A client program sends a flow of keywords, each randomly sampled from a pool of 50000 keywords, to the testing cluster for lookup. Each worker searches through its corresponding data block to find the required keyword and counts the number of lines containing the keyword. The line count is then sent back to the client program to sum up. The time to merge partial counts at the client program is negligible compared to the searching time. Therefore, this testing setup is similar to the framework we are considering in this chapter.

We measured the request response time, i.e., the time to process each keyword, at the client. We also collected the task response times at each worker. Note that the task response

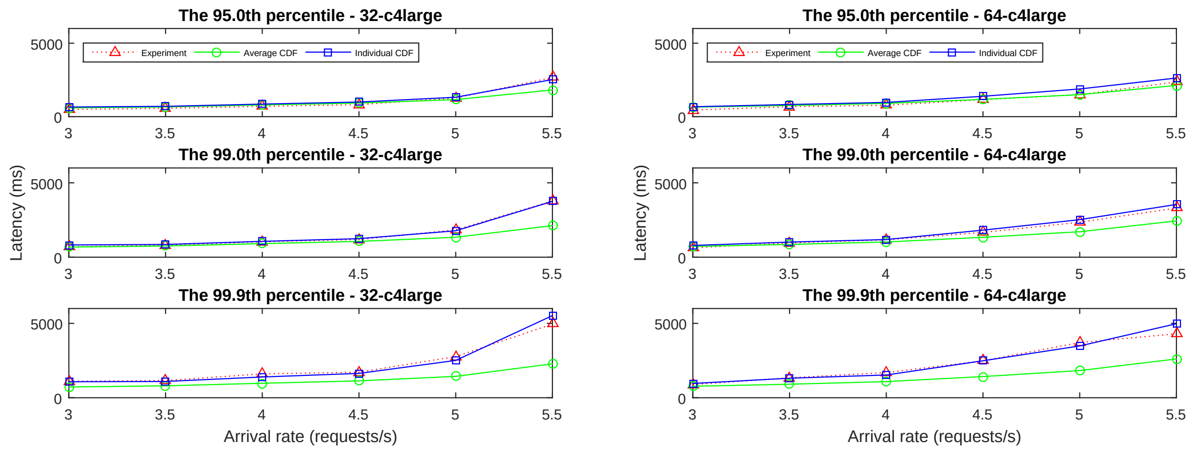


Figure 2.6: Predicted tail latencies for keyword occurrence counts in Amazon cloud with 32 (left) and 64 (right) nodes.

time is a summation of task waiting time and service time at each worker. The waiting time for a task is counted from the time the task's request is sent to the cluster to the time it is sent to a given worker for processing. The task service time is the actual processing time for the task at a given worker. From those collected samples, we computed the mean and variance of task response time, which were in turn used to derive the task response time distribution as in Eq. (2.1). The distributions from different workers might not be identical due to, e.g., uneven documents in data blocks or a bit variations in hardware, even with the same type of EC2 instances. Therefore, we calculated the request response time distribution using the inhomogeneous case given in Eq. (2.4), i.e., a product of individual distributions from the workers. We also compared the results with the assumption that the task response time distributions from all the workers are identical by computing the distribution from mean and variance of task response times across all the workers.

Figs. 2.6 shows a comparison of the latencies predicted from our proposed model against those from the experiments. The inhomogeneous model (the blue lines) gives a quite accurate

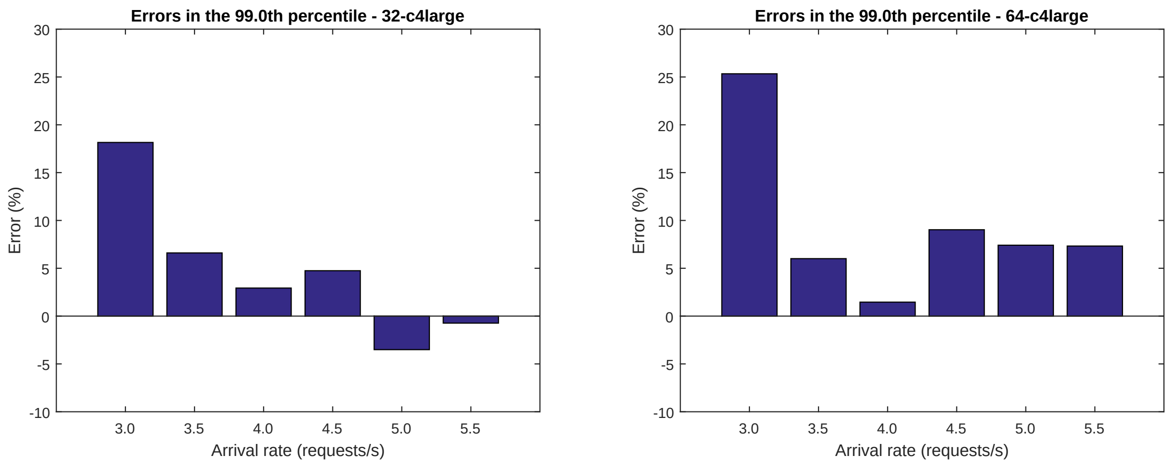


Figure 2.7: Prediction errors for cloud testbeds with 32 (left) and 64 (right) nodes.

prediction for the 95th, 99th, and 99.9th percentiles in both cluster settings, 32 and 64 worker nodes, while the homogeneous model (the green lines) only provides a good approximation for the 95th percentile and becomes worse when the cluster is under high load condition. The approximation errors for the 99th are within 10% when the system is at high loads as shown in Fig. 2.7. Note that we used the arrival rates on the horizontal axis based on the rates we sent requests to the testing cluster from the client program. We also measured the actual task processing time, i.e., task service time, on each worker while running experiments. Since the task service time distributions are likely not homogeneous, we estimated the equivalent loads for those arrival rates based on the maximum value of mean task service times across all the workers as given in Table. 2.1. One can see that, similar to the theoretical results, the prediction is more accurate when the cluster is at high loads.

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%

2.4 A General Model for Tail Latency Prediction

In the above sections, we considered the cases where an incoming job is forked to all the subsystems in the system. Here we look at a general case in which an incoming job is forked to only k random subsystems ($k \leq n$) out of n parallel subsystems in the system. This model is suitable for modeling distributed key-value systems in which a key lookup may touch only a partial number of servers, or web rendering which requires to receive web objects or data from multiple servers in a cluster.

Let X and Y be random variables for job response time and number of forked tasks per job with corresponding probability density functions $f_X(x)$ and $f_Y(y)$, respectively. The joint probability density function (PDF) of these two variables is given by

$$f_{XY}(x, y) = f_{X|Y}(x|y) f_Y(y) \tag{2.7}$$

where $f_{X|Y}(x|y)$ is the conditional PDF of job response time given a number of forked tasks per job, which is modeled as a generalized exponential distribution in this chapter.

In practice, one can assume that the number of forked tasks per job follows a discrete distribution and is in an integer range. Therefore, we have,

$$f_{XY}(x, y) = f_{X|Y}(x|y) P(Y = y) \tag{2.8}$$

It is easy to see that $f_{XY}(x, y)$ satisfies probability distribution properties, i.e., $f_{XY}(x, y) \geq$

0 for all x, y and

$$\begin{aligned}\int_x \int_y f_{XY}(x,y)dxdy &= \sum_y \int_x f_{X|Y}(x|y)P(Y=y)dx \\ &= \sum_y P(Y=y) \quad (\text{since } \int_x f_{X|Y}(x|y)dx = 1) \\ &= 1\end{aligned}$$

The marginal PDF with respect to X, i.e., job response time, is defined to be

$$f_X(x) = \sum_y f_{X|Y}(x|y)P(Y=y) \tag{2.9}$$

Similarly, the corresponding marginal cumulative distribution function (CDF) is given by

$$F_X(x) = \sum_y F_{X|Y}(x|y)P(Y=y) \tag{2.10}$$

We further assume that the numbers of forked tasks per job are k_i's, $i = 1, \ldots, m$, in a finite support $[a, b]$ with corresponding probabilities P_i's. Correspondingly, we have

$$F_X(x) = \sum_{i=1}^{m} P_i \cdot F_{X|Y}(x|k_i) \tag{2.11}$$

In this general model, the effective arrival rate at each queuing subsystem reduces by a factor of k/n. Therefore, to keep the same utilization at each subsystem as the above cases, we increase the system arrival rate λ by a factor of n/k in the experiments.

In the following sections, we assess the accuracy of our general model for the scenarios where the number of tasks is fixed, uniformly distributed, and triangularly distributed. We show the experimental results for the non-replicated systems with 100 and 1000 nodes with three different service time distributions, including exponential, truncated Pareto, and empirical distribution.

2.4.1 Scenario 1: Fixed Number

In this case, we consider the cases when the number of forked tasks per job is a fixed number k ($k \leq n$), i.e., every incoming job is dispatched to k random nodes in an n-node system.

From Eqs. (2.5) and (2.11), we have

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

and its mean and the pth percentile are given by

$$\begin{aligned} \mathbb{E}[X] &= \beta[\psi(k\alpha + 1) - \psi(1)]\,, & (2.13) \\ x_p &= -\beta \log\left(1 - \left(\frac{p}{100}\right)^{1/k\alpha}\right) & (2.14) \end{aligned}$$

Fig. 2.8 shows relative errors in the approximation of the 99th percentile for this scenario. Again, the model provides an accurate prediction at high load conditions

2.4.2 Scenario 2: Discrete Uniform Distribution

Here we deal with the cases when an incoming job is forked to k random nodes in the system where k is randomly sampled from an integer range $[a, b]$, i.e., $k_i \in [a, b]$ with probability $P_i = P\,\forall i$.

From Eqs. (2.5) and (2.11), we have

$$F_X(x) = P \cdot \sum_{i=1}^{m} (1 - e^{-x/\beta})^{k_i\alpha} \tag{2.15}$$

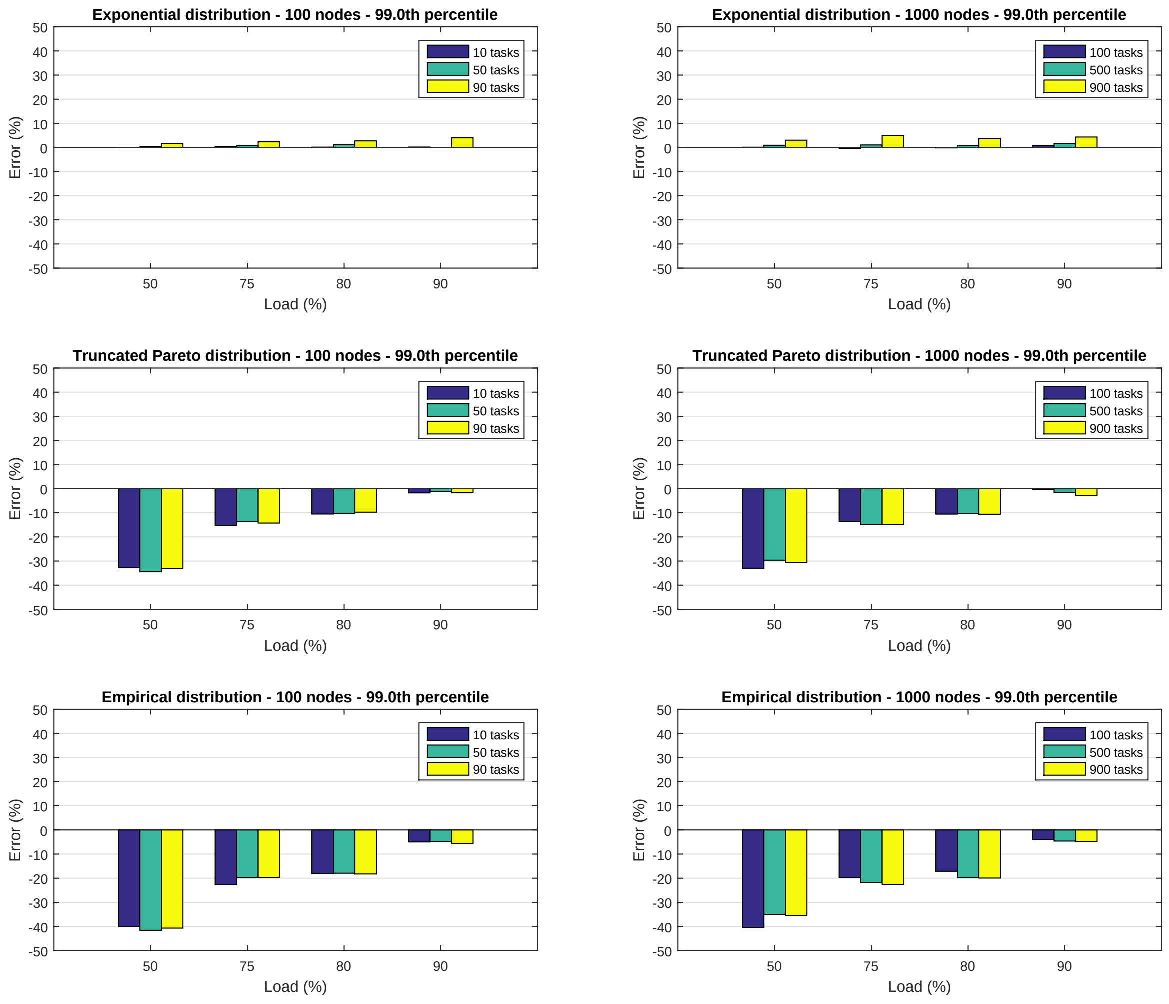


Figure 2.8: Errors in the 99th percentile prediction when the number of tasks per job is fixed.

and its mean and the pth percentile are given by

$$\mathbb{E}[X] = P \cdot \sum_{i=1}^{m} \beta[\psi(k_i\alpha + 1) - \psi(1)] \,, \tag{2.16}$$

$$x_p = F_X^{-1}(p/100) \,, \tag{2.17}$$

where $F_X^{-1}(.)$ is the inverse CDF of random variable X.

The experimental results provided in Fig. 2.9 again show the accuracy of the proposed model for the system under heavy traffic, an interest region of resource provisioning.

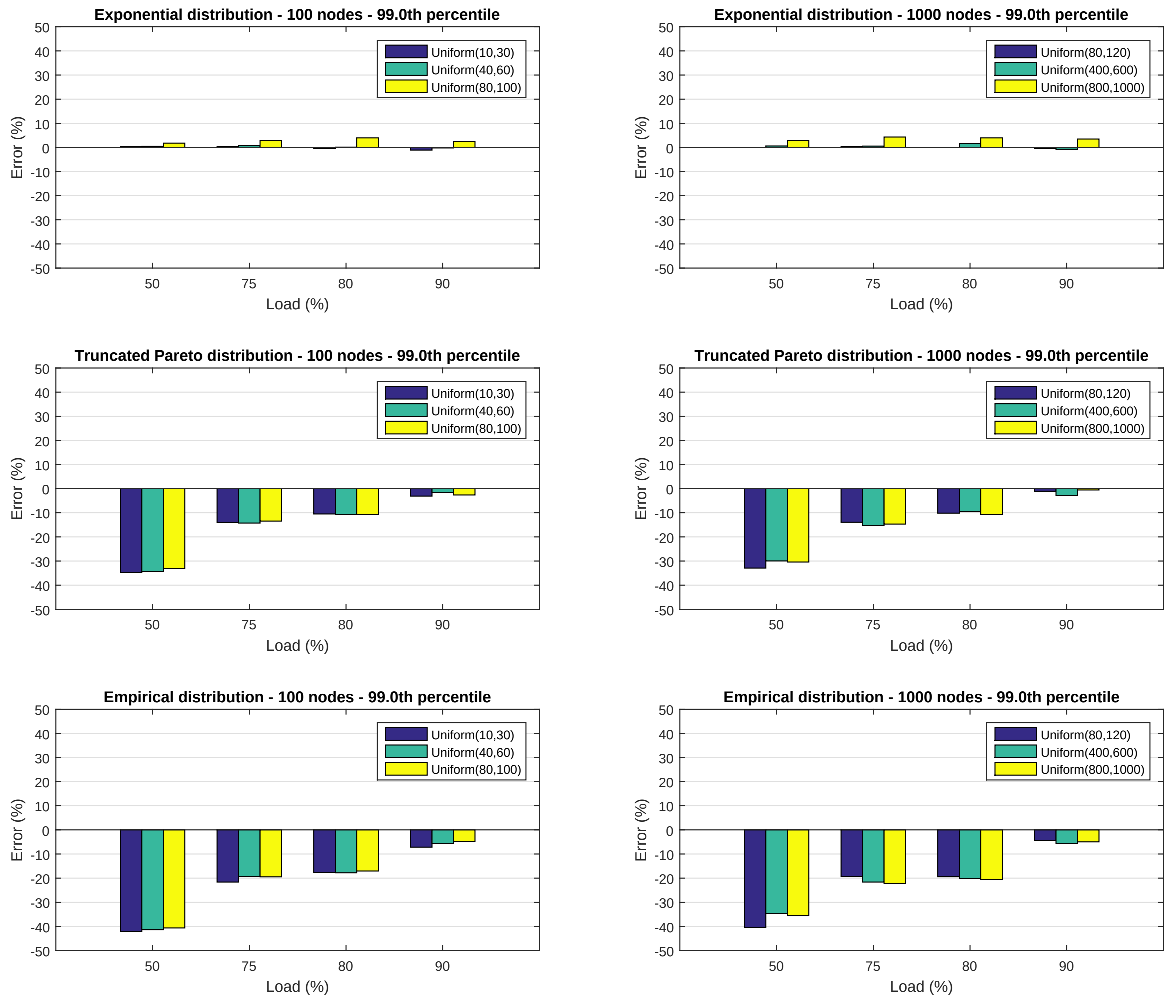


Figure 2.9: Errors in the 99th percentile prediction when the number of tasks per job is uniformly distributed.

2.4.3 Scenario 3: Discrete Triangular Distribution

In this scenario, we consider the case in which probability of number of forked tasks per job may vary. In particular, we assume it follows a discrete symmetrical triangular distribution [3] as in Fig. 2.10 with an integer support, i.e., $k_i \in [a, b]$ with probability

$P_i = P(k_i)$ for $i = 1, \ldots, m$,

$$P(k_i) \;=\; \begin{cases} d_l(k_i - a + 1) & a \le k_i \le c\,, \\ d_r(b - k_i + 1) & c < k_i \le b\,, \end{cases} \tag{2.18}$$

where c is the mode of the distribution, d_l and d_r are chosen so that

$$\sum_{k_i=a}^{b} P(k_i) = 1 \text{ and } d_l(c - a + 1) = d_r(b - c + 1) \tag{2.19}$$

For the symmetrical triangular distribution, i.e., $c - a = b - c$, given a, b, and c we can derive

$$d_l = d_r = d = \frac{1}{(c - a + 1)^2} \tag{2.20}$$

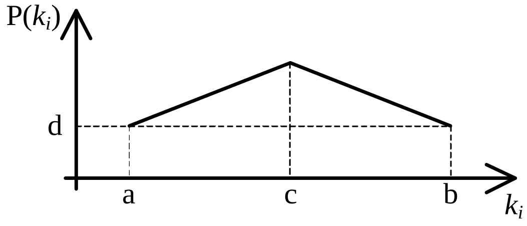


Figure 2.10: A discrete symmetrical triangular distribution.

From Eqs. (2.5) and (2.11), we have

$$F_X(x) = \sum_{i=1}^{m} P(k_i) \cdot (1 - e^{-x/\beta})^{k_i \alpha}\,, \tag{2.21}$$

where $P(k_i)$ is calculated from Eq. (2.18).

The mean and the pth percentile job response time are given by

$$\mathbb{E}[X] \;=\; \sum_{i=1}^{m} P(k_i) \cdot \beta[\psi(k_i\alpha + 1) - \psi(1)]\,, \tag{2.22}$$

$$x_p \;=\; F_X^{-1}(p/100) \tag{2.23}$$

In line with the previous case studies, the experimental results here show an accurate prediction for the 99th percentile at high load regions.

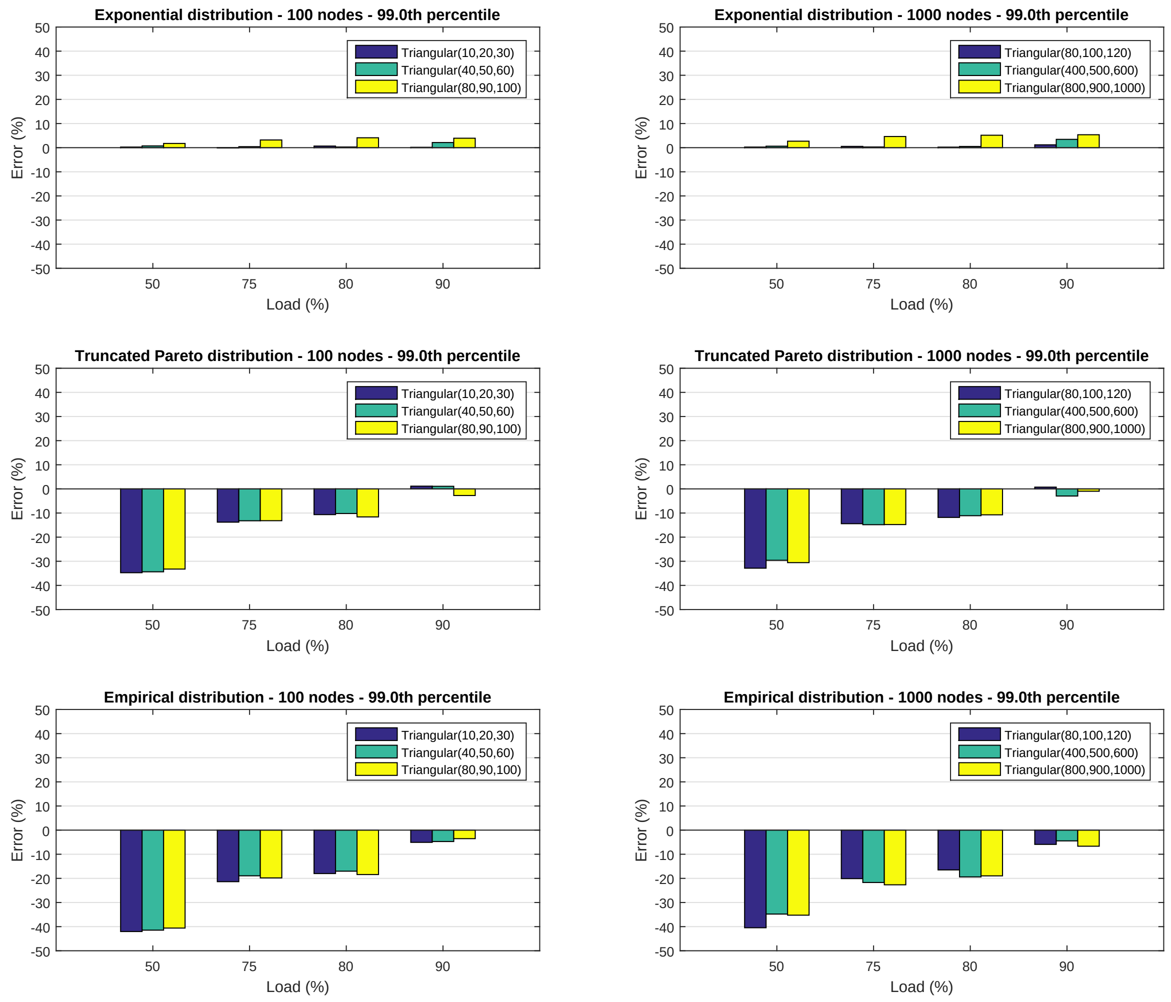


Figure 2.11: Errors in the 99th percentile prediction when the number of tasks per job is triangularly distributed.

2.5 Facilitating Resource Provisioning

In this section, we discuss how the above prediction models may be used to facilitate highly scalable, explicitly tail-constrained resource provisioning. For ease of discussion, we use the following example scenario as a guide throughout the discussion. Assume that a content service provider wants to outsource its OLDI scale-out services to a cloud service provider. With a given size of parallel searchable database D (e.g., an entire index as in a Web search engine) and monetary budget C, the content service provider wants to know whether or not the service to be deployed may sustain R requests per second, while meeting the tail SLO, i.e., the pth-percentile request response time of L ms.

An ad hoc approach is to immediately deploy the service to a certain scale and at run-time, scale out/up or down the system dynamically in a pay-as-you-go manner to meet the performance targets or monetary budget. Without an initial estimation, however, such approaches run the risk of either over budgeting or failing to meet SLO and/or targeted request throughput performance. Moreover, using the pay-as-you-go service for dynamic resource provisioning is generally much more expensive than static resource reservation for resource planning [10]. Given the sheer size of the system to be deployed, it is of paramount importance to develop an offline, highly scalable resource provisioning approach that can provide a quick initial assessment of whether the performance targets and monetary budget can be met or not.

The idea of our approach is sketched by the following tail-constrained resource provisioning procedure, in the context of the above example scenario:

– For a desired type of VMs with, e.g., given CPU speed, memory size, and pricing model, build a replicated server cluster subsystem in the cloud using m (two to three) VMs by replicating a portion of the total database, i.e., D/N, to all the VM replicas, where N, an integer value, may be selected in such a way that D/N can fit comfortably in the memory in each VM;

– Measure the mean and variance of task response time in the cluster running a task scheduling policy, at desired task rate $\lambda = R$;

- Find the parameters of the generalized exponential distribution in Eq. (2.1) by plugging in the measured mean and variance task latency into Eqs. (2.2) and (2.3), respectively;
- Estimate the pth-percentile request response time x_p based on Eq. (2.6);
- Finally, x_p is compared against L and the total cost for running N VM clusters with m each is compared against the associated budget C, to see if both the tail SLO and monetary budget are met. If both are met, a feasible tail-constrained resource provisioning is found. Otherwise, the performance targets and/or budget are revised and then rerun the procedure. Note that if x_p is found well below L, one may consider reducing N and/or m and see if it is still below L. This iterative testing can help minimize the cost.

2.6 Conclusions

This chapter proposed elegant prediction models for the tail SLOs of scale-out applications involving task-partitioning-merging. The required inputs to the prediction models are only the mean and variance of task response times for a task mapped to a subsystem. The experimental results showed that the proposed models yield accurate predictions with errors consistently within 10% at the server loads of 90% or higher, providing a much needed prediction tool to facilitate tail-constrained resource provisioning for scale-out applications.

Chapter 3

TAIL AND MEAN LATENCY PREDICTION FOR FORK-JOIN QUEUING NETWORKS

3.1 Introduction

The Fork-Join queuing networks have been studied extensively in the literature. To date, the exact solution exists for a two-way network only [25, 45]. Most of work primarily focus on the approximation of mean response time [6, 38, 45, 58] and its bounds [8, 18]. Several works dealt with the approximation of response time distribution assuming a simple queuing model for each forked node, e.g., M/M/1 [9] or M/M/k [36]. For networks with general service time distribution, several works have introduced hybrid approaches that combine analysis and simulation to derive the approximation for mean response time [17, 45, 56]. Please refer to [55] and references therein for a detailed survey on Fork-Join queuing networks. In the following, we highlight several analytical and hybrid approaches for mean response time approximation.

Nelson and Tantawi [45] introduced a hybrid approach for the approximation of mean response time, R_n, of the Fork-Join network with n M/M/1 queues ($2 \leq n \leq 32$) based on the exact solution for the two-way network [25],

$$R_n \approx \left[\frac{\mathcal{H}_n}{\mathcal{H}_2} + \left(1 - \frac{\mathcal{H}_n}{\mathcal{H}_2}\right)\frac{4\rho}{11}\right]\left(\frac{12-\rho}{8}\right)(\mu-\lambda)^{-1}\,, n \geq 2\,, \tag{3.1}$$

where μ is the mean service rate of each queue, λ is the arrival rate, $\rho = \lambda/\mu$ is the utilization on each queue, and $\mathcal{H}_n$ is the nth harmonic number, $\mathcal{H}_n = \sum_{i=1}^{n} \frac{1}{i}$.

Another approximation for the Fork-Join network with M/M/1 queues was derived based on light and heavy traffic condition [59],

$$R_n \approx \left[\mathcal{H}_n + \left(\left(\sum_{i=1}^{n}\binom{n}{i}(-1)^{i-1}\sum_{m=1}^{i}\binom{i}{m}\frac{(m-1)!}{i^{m+1}}\right) - \mathcal{H}_n\right)\frac{\lambda}{\mu}\right](\mu-\lambda)^{-1}\,, \tag{3.2}$$

In [4], the authors presented an approximation based on the response time optimistic and pessimistic bounds,

$$R_n \approx \left[\mathcal{H}_n + \frac{\rho}{2(1-\rho)}\left(\sum_{i=1}^{n}\frac{1}{i-\rho} + (1-2\rho)\sum_{i=1}^{n}\frac{1}{i(i-\rho)}\right)\right]\frac{1}{\mu}\,. \tag{3.3}$$

In a recent work [38], a simulation study has assessed the accuracy of the approximation based on order statistic [57]. For M/M/1 queues, it showed that the approach is overestimating and worse than those from Eqs. (3.1)–(3.3). The approach works better for the queues with general service time distribution, i.e., M/G/1 queues.

In this chapter, we present an analytical approach for prediction of tail and mean latency for Fork-Join queuing networks with arbitrary service time distributions for the fork nodes. We first formalize the prediction model for tail latency using M/G/1 queuing model for each subsystem. We then propose an empirical method for the approximation of mean latency based on the tail latency prediction and observations from the ratios between tail (e.g., the 99th percentile) and mean latency. These approximations will allow scaling properties of scale-out data-intensive applications to be studied, as we shall show in the next chapter.

3.2 The Analytical Model for Tail Latency Prediction

Here we consider a Fork-Join queuing network as in the previous chapter but each subsystem is now viewed as an M/G/1 queuing model as in Fig. 3.1. We assume that the job arrival is a Poisson process with mean arrival rate λ and the service times follow a general distribution to capture different underlying subsystems. Now our goal is to find the job response time distribution for the Fork-Join queuing model in Fig. 3.1. Finding this distribution will then allow tail and mean job response times to be calculated. To this end, we first need to find the task response time distribution for each Fork node in Fig. 3.1, denoted as $F_{X_i}(x_i)$ for $i = 1, ..., n$. Then assuming X_i's are independent random variables, the job response time distribution is given as

$$F_Y(y) = \prod_{i=1}^{n} F_{X_i}(x_i) \tag{3.4}$$

So now the key is to find $F_{X_i}(x_i)$.

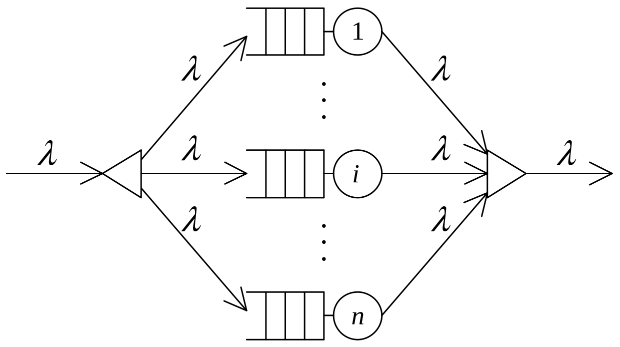


Figure 3.1: A system with n subsytems each viewed as an M/G/1 queuing model.

Consider a single subsystem, modeled as an M/G/1 queuing model. Let W, S, and X denote its waiting time, service time, and response time random variables, respectively. We have,

$$X = W + S$$

since W and S are independent and mean and variance of the response time are thus given as

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

From Takács recurrence theorem [35], 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 following relations can be derived

$$\begin{aligned}
\mathbb{E}[W] &= \frac{\lambda\mathbb{E}[S^2]}{2(1-\rho)} = \frac{\rho\mathbb{E}[S]}{1-\rho}\left(\frac{1+C_S^2}{2}\right), \\
\mathbb{E}[W^2] &= 2\mathbb{E}[W]^2 + \frac{\lambda\mathbb{E}[S^3]}{3(1-\rho)}\,, \\
\mathbb{V}[W] &= \mathbb{E}[W^2] - \mathbb{E}[W]^2 = \mathbb{E}[W]^2 + \frac{\lambda\mathbb{E}[S^3]}{3(1-\rho)}\,, \\
\mathbb{E}[X] &= \mathbb{E}[W] + \mathbb{E}[S] = \mathbb{E}[S]\left(1 + \frac{\rho}{1-\rho}\cdot\frac{1+C_S^2}{2}\right), \qquad (3.5) \\
\mathbb{V}[X] &= \mathbb{V}[W] + \mathbb{V}[S] = \mathbb{E}[W]^2 + \frac{\lambda\mathbb{E}[S^3]}{3(1-\rho)} + \mathbb{E}[S^2] - \mathbb{E}[S]^2, \qquad (3.6)
\end{aligned}$$

where $\mathbb{E}[S^k]$ the kth moment of the service time and $C_S^2 = \mathbb{V}[S]/\mathbb{E}[S]^2$ is the squared coefficient of variation of service times.

In principle, the response time distribution can be derived by using inverse Laplace transform but it is generally difficult to find the exact closed-form solutions. We propose to approximate the task response time distribution using a generalized exponential distribution

[27] similar to the approximation used in the previous chapter,

$$F_X(x) = (1 - e^{-x/\beta})^{\alpha} \, , \tag{3.7}$$

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

The mean and variance of this distribution are given by

$$\begin{aligned} \mathbb{E}[X] &= \beta[\psi(\alpha+1) - \psi(1)] \, , & (3.8) \\ \mathbb{V}[X] &= \beta^2[\psi'(1) - \psi'(\alpha+1)] \, , & (3.9) \end{aligned}$$

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

The CDF of job response time Y of a homogeneous n-node network can then be written as,

$$F_Y(y) = (F_X(x))^n = (1 - e^{-y/\beta})^{n\alpha} \tag{3.10}$$

and its mean and variance are given to be

$$\begin{aligned} \mathbb{E}[Y] &= \beta[\psi(n\alpha+1) - \psi(1)] \, , & (3.11) \\ \mathbb{V}[Y] &= \beta^2[\psi'(1) - \psi'(n\alpha+1)] \, , & (3.12) \end{aligned}$$

The pth percentile of the response time can be derived from Eq. (3.10) as

$$y_p = -\beta \log\left(1 - \left(\frac{p}{100}\right)^{1/n\alpha}\right) \tag{3.13}$$

The approach taken here differs from the one used in the previous chapter in that it requires to collect only service time samples. This can be done by sending jobs to a testing subsystem one by one and collecting the processing time for each job. In this way, the approach does not depend on the traffic pattern of the job flow coming to the system.

In summary, the steps to obtain the pth percentile of job response times for a system with a given task service time distribution are as follows,

– Compute the mean, variance, and third moment for the given task service time distribution or empirical data.

– Find the mean and variance of task response times from Eqs. (3.5) and (3.6)

– Substitute the above mean and variance into (3.8) and (3.9) and solve that system of equations to find the scale and shape parameters of the generalized exponential distribution used to approximate the task response time distribution.

– Calculate mean, variance, and the pth percentile of job response times from Eqs. (3.11), (3.12), and (3.13), respectively.

3.3 Evaluation of the Tail Latency Prediction Model

In this section, we test the proposed approximation model for tail latency with different service time distributions, including light-tailed, heavy-tailed, and empirical distributions, as in chapter 2. The mean service time is set to 4.22ms, which is based on the empirical distribution, for all the distributions under study. Specifically, the following distributions are considered:

– A light-tailed exponential distribution ($CV = 1.0$) (scenario 1)

– A light-tailed Gamma distribution ($CV = 0.8$) (scenario 2)

– A heavy-tailed truncated Pareto distribution ($CV = 1.20$) (scenario 3)

– A heavy-tailed Weibull distribution ($CV = 1.50$) (scenario 4)

– A heavy-tailed Empirical distribution measured from a Google search test leaf node ($CV = 1.12$) (scenario 5)

For all the cases, we show the relative error of the predicted result, t_{pred}, against the simulation one, t_{sim}, i.e.,

$$error = \frac{100(t_{pred} - t_{sim})}{t_{sim}} .$$

3.3.1 Scenario 1: Exponential Distribution

The CDF is defined as [13]

$$F_S(s) = 1 - e^{-\mu s} , \tag{3.14}$$

where μ is the mean service rate ($\mu > 0$).

The response time distribution is a closed form, which is given by [13]

$$F_X(x) = (1 - e^{-\mu(1-\rho)x}) , \tag{3.15}$$

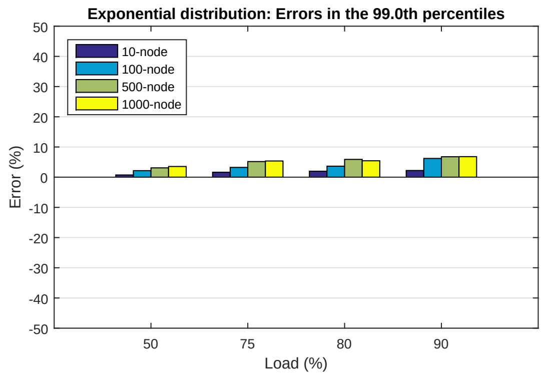
i.e., for this case, $\alpha = 1$ and $\beta = 1/\mu(1-\rho)$ when compared with Eq. (3.7).

Substituting into Eqs. (3.11) and (3.13), we have,

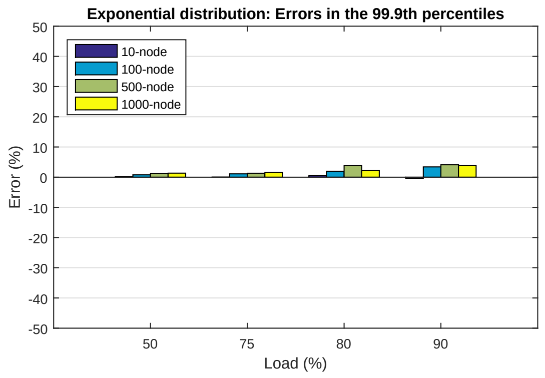
$$\begin{aligned} \mathbb{E}[Y] &= \frac{1}{\mu(1-\rho)}[\psi(n+1) - \psi(1)] = \frac{1}{\mu(1-\rho)}\mathcal{H}_n , && (3.16) \\ y_p &= -\frac{1}{\mu(1-\rho)} \log\left(1 - \left(\frac{p}{100}\right)^{1/n}\right) , && (3.17) \end{aligned}$$

where $\mathcal{H}_n$ is the nth harmonic number, $\mathcal{H}_n = \sum_{k=1}^{n} \frac{1}{k}$.

Fig. 3.2 shows the relative errors of the 99th and 99.9th percentiles calculated from the prediction model to the ones from simulation. The model yields a very good approximation for the tail latencies for all the cases under study, with all the errors being within 7%.



(a) The 99th percentiles



(b) The 99.9th percentiles

Figure 3.2: Exponential distribution: Errors in the 99th and 99.9th percentiles.

3.3.2 Scenario 2: Gamma Distribution

Here we consider another light-tailed case, Gamma distribution with mean service time $m = 1.0$ and coefficient of variation $C_S = 0.8$. The CDF and PDF of the Gamma distribution are defined as

$$\text{CDF:} \quad F_S(s) = \frac{1}{\Gamma(a)}\gamma(a, \frac{s}{b}) \,, \tag{3.18}$$

$$\text{PDF:} \quad f_S(s) = \frac{1}{b\Gamma(a)}\left(\frac{s}{b}\right)^{a-1} e^{-s/b} \,, \tag{3.19}$$

where $\Gamma(.)$ is a Gamma function and a and b are shape and scale parameters, respectively.

The kth moment is given by

$$\mathbb{E}[S^k] = b^k \cdot \frac{\Gamma(a+k)}{\Gamma(a)} \tag{3.20}$$

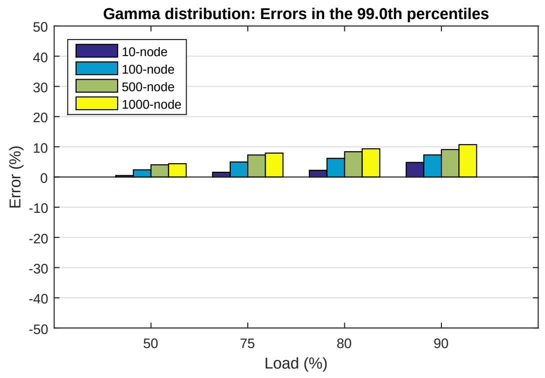
From Eq. (3.20), the following relations can be derived

$$\mathbb{E}[S] = ba \,, \tag{3.21}$$

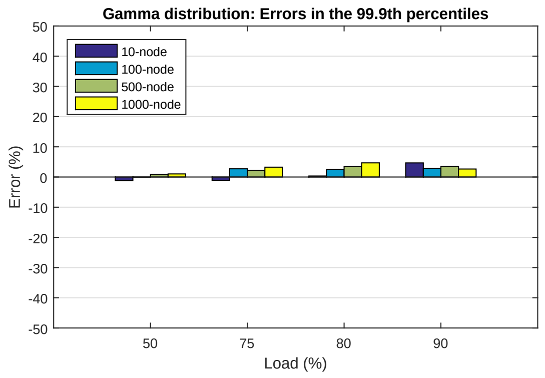
$$\mathbb{V}[S] = b^2 a \,, \tag{3.22}$$

$$\mathbb{E}[S^3] = b^3 \cdot \frac{\Gamma(a+3)}{\Gamma(a)} = b^3(a+2)(a+1)a \,. \tag{3.23}$$

The prediction model yields a good approximation for tail latencies, the 99th and 99.9th percentiles, as shown in Fig. 3.3.



(a) The 99th percentiles



(b) The 99.9th percentiles

Figure 3.3: Gamma distribution: Errors in the 99th and 99.9th percentiles.

3.3.3 Scenario 3: Truncated Pareto Distribution

The CDF and PDF of this distribution are given by [5]

$$\text{CDF:} \quad F_S(s) = \begin{cases} \dfrac{1-(L/s)^a}{1-(L/H)^a} & L \le s \le H, \\ 0 & \text{otherwise}, \end{cases} \tag{3.24}$$

$$\text{PDF:} \quad f_S(s) = \begin{cases} \dfrac{aL^a s^{-a-1}}{1-(L/H)^a} & L \le s \le H, \\ 0 & \text{otherwise}, \end{cases} \tag{3.25}$$

where α is shape parameter, L and H are lower and upper bound, respectively.

The kth moment can be derived as

$$\begin{aligned} \mathbb{E}[S^k] &= \int_{-\infty}^{\infty} s^k f_S(s) ds = \frac{aL^a}{1-(L/H)^a} \int_L^H s^{k-a-1} ds \,, \\ &= \frac{L^a}{1-(L/H)^a} \cdot \frac{a}{a-k} \left(\frac{1}{L^{a-k}} - \frac{1}{H^{a-k}} \right) \quad \text{for } a \neq k \end{aligned} \tag{3.26}$$

Therefore, we have,

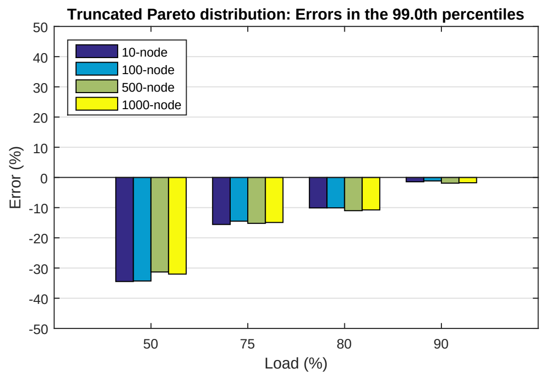
$$\mathbb{E}[S] = \frac{L^a}{1-(L/H)^a} \cdot \frac{a}{a-1}\left(\frac{1}{L^{a-1}} - \frac{1}{H^{a-1}}\right) \qquad a \neq 1\,, \tag{3.27}$$

$$\mathbb{E}[S^2] = \frac{L^a}{1-(L/H)^a} \cdot \frac{a}{a-2}\left(\frac{1}{L^{a-2}} - \frac{1}{H^{a-2}}\right) \qquad a \neq 2\,, \tag{3.28}$$

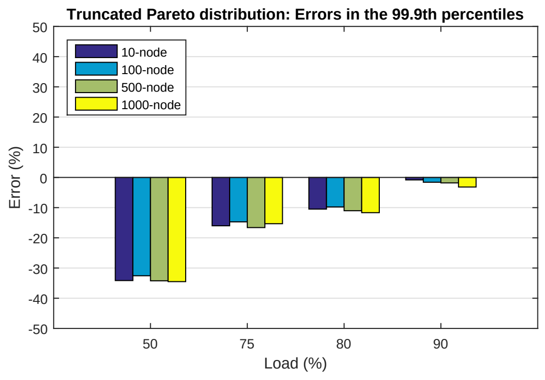
$$\mathbb{V}[S] = \mathbb{E}[S^2] - (\mathbb{E}[S])^2\,, \tag{3.29}$$

$$\mathbb{E}[S^3] = \frac{L^a}{1-(L/H)^a} \cdot \frac{a}{a-3}\left(\frac{1}{L^{a-3}} - \frac{1}{H^{a-3}}\right) \qquad a \neq 3\,. \tag{3.30}$$

Fig. 3.4 shows the relative errors of the 99th and 99.9th of the prediction model. Similar to the case in chapter 2, the model gives a good approximation when the system is under heavy traffic, i.e., at load of 75% or more.



(a) The 99th percentiles



(b) The 99.9th percentiles

Figure 3.4: Truncated Pareto distribution: Errors in the 99th and 99.9th percentiles.

3.3.4 Scenario 4: Weibull Distribution

The CDF of Weibull distribution is defined as

$$F_S(s) = 1 - \exp[-(x/b)^a] \quad s \geq 0 \tag{3.31}$$

where a and b are shape and scale parameter, respectively.

The kth moment is given by

$$\mathbb{E}[S^k] = b^k \cdot \Gamma\left(1 + \frac{k}{a}\right) \tag{3.32}$$

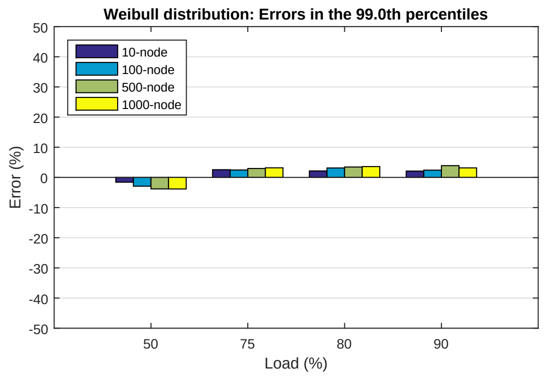
From Eq. (3.32), we can derive the following relations,

$$\mathbb{E}[S] = b \cdot \Gamma\left(1 + \frac{1}{a}\right), \tag{3.33}$$

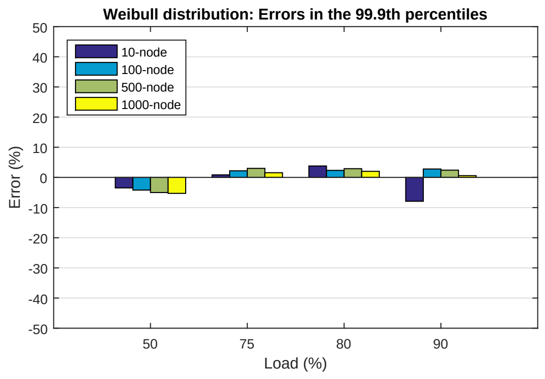
$$\mathbb{V}[S] = \mathbb{E}[S^2] - (\mathbb{E}[S])^2 = b^2 \cdot \left[\Gamma\left(1 + \frac{2}{a}\right) - \left(\Gamma\left(1 + \frac{1}{a}\right)\right)^2\right], \tag{3.34}$$

$$\mathbb{E}[S^3] = b^3 \cdot \Gamma\left(1 + \frac{3}{a}\right). \tag{3.35}$$

Fig. 3.5 shows the relative errors of the 99th and 99.9th percentiles of the prediction model. As in the previous chapter, the model provides an accurate prediction for tail latencies for the entire range of study.



(a) The 99th percentiles



(b) The 99.9th percentiles

Figure 3.5: Weibull distribution: Errors in the 99th and 99.9th percentiles.

3.3.5 Scenario 5: Empirical Distribution

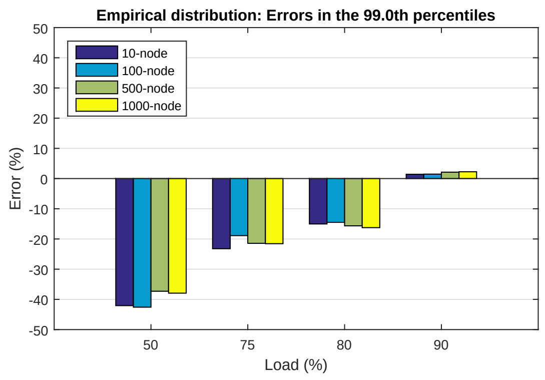
In this scenario, we consider the cases when one has only the empirical data collected from a test subsystem, i.e., the service time distribution is unknown. Here we approximate the first three moments of the underlying distribution using the corresponding sample moments.

Assume that there is a sequence of random variables, $S_1, S_2, \ldots, S_n$, corresponding to the empirical data values, $s_1, s_2, \ldots, s_n$. The kth sample moment, i.e., the kth moment about origin, can be drawn from those data as

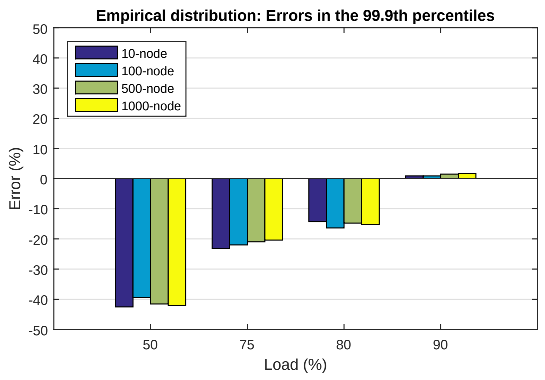
$$\mathbb{E}[S^k] = \frac{1}{n}\sum_{i=1}^{n} s_i^k \tag{3.36}$$

Given collected data samples, one can easily compute $\mathbb{E}[S]$, $\mathbb{V}[S] = \mathbb{E}[S^2] - \mathbb{E}[S]^2$, and $\mathbb{E}[S^3]$ from Eq. (3.36).

Fig. 3.6 shows the relative errors of the 99th and 99.9th percentiles calculated from the prediction model to the ones from simulation. Similar to the prediction in chapter 2, the model yields a good approximation when the system is under heavy load, e.g., 80% or 90%.



(a) The 99th percentiles



(b) The 99.9th percentiles

Figure 3.6: Empirical distribution: Errors in the 99th and 99.9th percentiles.

In summary, this analytical model yields accurate predictions for tail latencies, e.g., the 99th and 99.9th percentiles, in all the cases under study when the system is at high loads. This agrees with the results in chapter 2. More detailed results for this model are given in Appendix. A.

3.4 Approximation for Mean Latency

In this section, we present an empirical method for finding the approximation of mean response time or latency in Fork-Join queuing networks. The approach is based on the observation on ratios of tail and mean latency. From the experiments of the tail latency prediction, we found that the differences in those ratios between the ones from the prediction model and from the experiments are almost constant across all the number of nodes under study when the system is at high loads as shown in Tables. 3.1–3.5. As a result, the approximation of mean response time can be computed by utilizing the results from the prediction model for tail latency, e.g., the 99th percentile.

Let x_p^{ge} and m_p^{ge} are the pth percentile and mean response time calculated from Eqs. (3.13) and (3.11), respectively, and x_p and m_p are the predicted values of the pth percentile and mean response time, respectively. From the observations, we have,

$$\frac{x_p}{m_p} - \frac{x_p^{ge}}{m_p^{ge}} = \Delta \, . \tag{3.37}$$

Let $R_{ge} = x_p^{ge}/m_p^{ge}$. The mean response time can be predicted as follows,

$$m_p = \frac{x_p}{R_{ge} + \Delta} \, , \tag{3.38}$$

where $x_p \approx x_p^{ge}$ at high loads, Δ is the 99th/mean ratio difference between the experiment and the prediction model. So, if we know Δ, we have m_p. In other words, the key is how to find Δ.

From Tables. 3.1–3.5, we also observed that there is a clear connection between the distribution tail heaviness and the ratio differences, Δ's. The differences are larger for light-tailed distributions, e.g., Exponential or Gamma distribution, and smaller for heavy-tailed ones, e.g., truncated Pareto or empirical distribution. To put it differently, this suggests that the Δ value is more dependent on the heaviness of the tail of a service time distribution and much less on the actual distribution itself. This observation inspires us to further explore a possible universal relationship between Δ and the heaviness of the distribution's tail.

Table 3.1: Exponential distribution: Differences in the 99th and mean latency ratios

	Number of nodes									
Load	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
75%	0.488	0.489	0.479	0.487	0.484	0.486	0.497	0.487	0.479	0.492
80%	0.512	0.501	0.524	0.515	0.504	0.536	0.515	0.517	0.513	0.515
90%	0.538	0.534	0.593	0.569	0.551	0.553	0.570	0.568	0.558	0.565

Table 3.2: Gamma distribution: Differences in the 99th and mean latency ratios

	Number of nodes									
Load	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
75%	0.628	0.647	0.631	0.639	0.628	0.629	0.643	0.637	0.649	0.636
80%	0.660	0.670	0.674	0.676	0.657	0.685	0.670	0.685	0.668	0.665
90%	0.732	0.766	0.745	0.728	0.754	0.732	0.707	0.730	0.746	0.728

Table 3.3: Weibull distribution: Differences in the 99th and mean latency ratios

	Number of nodes									
Load	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
75%	0.239	0.237	0.233	0.237	0.235	0.233	0.230	0.239	0.230	0.234
80%	0.247	0.238	0.254	0.248	0.253	0.256	0.253	0.263	0.251	0.255
90%	0.288	0.278	0.302	0.299	0.280	0.323	0.301	0.287	0.286	0.308

To explore the relation between the ratio differences and service time distribution's tail heaviness, we further perform experiments on the truncated Pareto distribution with different coefficient of variations, i.e., different tail heavinesses as shown in Table. 3.6. The mean service time is set to 1.0 in all the experiments. Tables. 3.7 and 3.8 present the example

Table 3.4: Truncated Pareto distribution: Differences in the 99th and mean latency ratios

	Number of nodes									
Load	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
75%	0.162	0.110	0.091	0.080	0.085	0.078	0.078	0.084	0.080	0.077
80%	0.147	0.116	0.119	0.107	0.109	0.104	0.105	0.108	0.095	0.106
90%	0.193	0.187	0.185	0.185	0.206	0.175	0.186	0.180	0.179	0.199

Table 3.5: Empirical distribution: Differences in the 99th and mean latency ratios

	Number of nodes									
Load	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
75%	0.192	0.129	0.094	0.067	0.074	0.083	0.062	0.076	0.087	0.073
80%	0.180	0.112	0.109	0.099	0.093	0.092	0.097	0.098	0.091	0.102
90%	0.208	0.152	0.175	0.157	0.156	0.173	0.149	0.162	0.193	0.151

ratios calculated from the experiments with $CV = 3.5$ and $CV = 6.0$. From those tables, one can see that when the service time distribution becomes heavier, e.g., corresponding to $CV \geq 3.5$, the ratio differences becomes very small compared to the ratios themselves. This means that starting from that threshold, the mean latency can be estimated directly using the one from the tail latency prediction model given in Eq. (3.11), i.e., $\Delta \approx 0$.

Table 3.6: Truncated Pareto distribution: Differences in the 99th and mean latency ratios for different CVs.

	Coefficient of Variation (CV)								
Load	**1.5**	**2.0**	**2.5**	**3.0**	**3.5**	**4.0**	**4.5**	**5.0**	**6.0**
75%	0.045	0.001	-0.016	-0.006	-0.012	-0.013	-0.013	-0.011	-0.017
80%	0.047	0.019	0.010	0.005	-0.005	0.002	-0.004	-0.009	-0.003
90%	0.120	0.065	0.048	0.046	0.035	0.003	0.004	0.012	-0.008

From all those observations, we propose an empirical approach to mean latency approximation based on the tail heaviness of a given service time distribution. To measure the tail heaviness, or equivalently, tail weight, we use Right Quantile Weight [15] which mea-

Table 3.7: Truncated Pareto distribution: The 99th and mean latency ratios at $CV = 3.5$

	Number of nodes					
Load	**100**	**200**	**400**	**600**	**800**	**1000**
75%	1.905	1.799	1.712	1.664	1.639	1.600
80%	1.877	1.784	1.694	1.657	1.619	1.607
90%	1.900	1.770	1.688	1.653	1.629	1.587

Table 3.8: Truncated Pareto distribution: The 99th and mean latency ratios at $CV = 6.0$

	Number of nodes					
Load	**100**	**200**	**400**	**600**	**800**	**1000**
75%	1.807	1.733	1.681	1.622	1.606	1.559
80%	1.845	1.707	1.678	1.596	1.602	1.581
90%	1.783	1.694	1.652	1.617	1.509	1.575

sures the tail heaviness on the right side of a distribution, the region of interest in all of our experiments. This tail weight measure is defined to be

$$\tau(F) = \frac{F^{-1}((1+q)/2) + F^{-1}((1-q)/2) - 2F^{-1}(0.75)}{F^{-1}((1+q)/2) - F^{-1}((1-q)/2)}\ , \tag{3.39}$$

where $0.5 < q < 1$ and $F^{-1}(q)$ is quantile q of distribution F.

There is no optimal choice for q. The tail weight measure is less robust to outliers if the selected value of q is too large, i.e., close to 1. Since we are studying the Fork-Join model where the slowest task response time determines its job response time, the extreme values of the service time distribution has a significant effect on the results. To capture the tail effect but still retain a reasonable robustness, we select $q = 0.99$.

We now can build a two-dimension lookup table whose coordinates are tail weights and utilizations, which should put the subsystems under heavy traffic to achieve more accurate approximation. The more entries we have in the table the better approximation we can achieve. However, for the purpose of evaluating the approximation, we leave out the distributions in the tail latency prediction above as testing cases. That is, we build the lookup table whose entries include only measures from exponential and truncated Pareto distributions in Table. 3.6 to capture both light-tailed and heavy-tailed distributions. An example of the lookup table is given in Table. 3.9. Theoretically, one can build the table at a finer degree, i.e., with as many entries as possible, to achieve better approximation.

Table 3.9: An example lookup table for differences in the 99th percentile and mean latency ratios

	Tail weight									
Load	**0.703**	**0.918**	**0.935**	**0.949**	**0.960**	**0.969**	**0.977**	**0.983**	**0.989**	**0.997**
75%	0.486	0.045	0.001	-0.016	-0.006	-0.012	-0.013	-0.013	-0.011	-0.017
80%	0.510	0.047	0.019	0.010	0.005	-0.005	0.002	-0.004	-0.009	-0.003
90%	0.575	0.120	0.065	0.048	0.046	0.035	0.003	0.004	0.012	-0.008

Steps to find the mean response time can be summarized as follows,

– Compute the 99th percentile and the ratio of the 99th percentile and mean latency based on the approximation from the generalized exponential distribution in Eqs. (3.13) and (3.11).

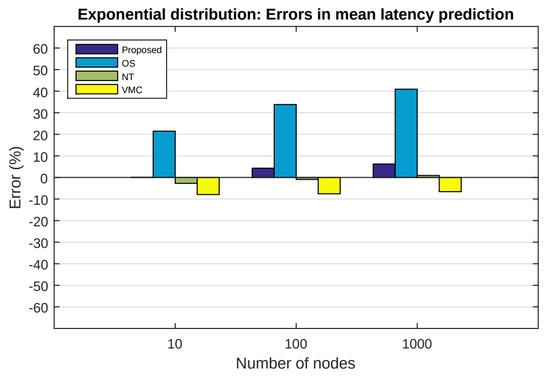
– With a given service time distribution and a desired load, calculate the distribution tail weight from Eq. (3.39) and then select the closest entry or interpolate the 99th/mean ratio difference from the lookup table.

– Approximate the mean response time using Eq. (3.38).

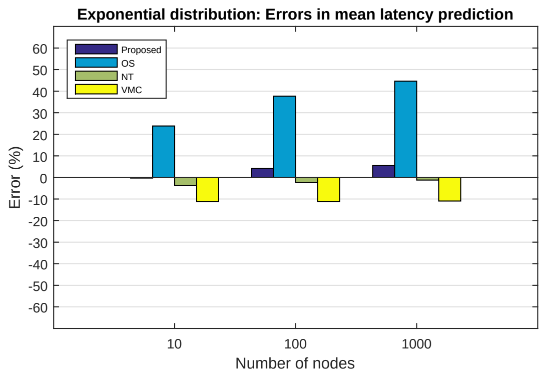
3.4.1 Scenario 1: M/M/1 Queues

The M/M/1 queuing model for each subsystem is often considered in analytical methods for the approximation of mean response time of Fork-Join queuing networks [4, 25, 45, 59].

In this section, we compare our proposed approximation with the ones from Eq. (3.1) [45] (named NT) and Eq. (3.3) [4] (named VMC). We also assess the accuracy of the direct approximation based on the order statistic (named OS), i.e., as in Eq. (3.11). Due to numerical issues with large n's, we did not include the approximation from Eq. (3.2) in the comparison. With small n's, it is a little better than the one in Eq. (3.3) but not as good as the approximation in Eq. (3.1).



(a) Load of 80%



(b) Load of 90%

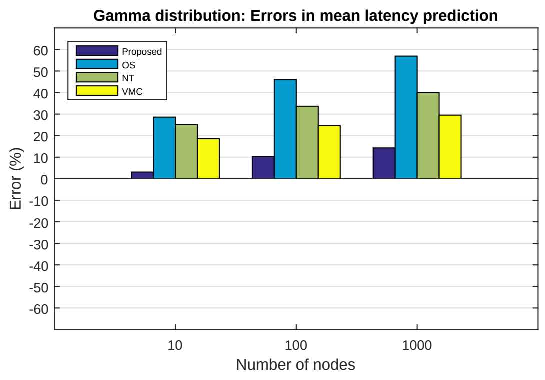
Figure 3.7: Exponential distribution: Comparison of mean response time approximation techniques.

Fig. 3.7 shows the comparison with $n = 10, 100, 1000$ at utilizations of 80% and 90%. The approximation in Eq. (3.1) gives the best accuracy in this case study. The proposed approach yield errors within 6.3% for all the cases studied and its accuracy is close to that in Eq. (3.1). However, as we shall see, the approximation in Eq. (3.1) could not be applied to general service time distributions other than exponential distribution. The direct approximation based on the order statistic, i.e., without ratio offsets as in the proposed approach, has the worst accuracy among all the cases under study. In this case, it is the upper bound for the mean response time [55].

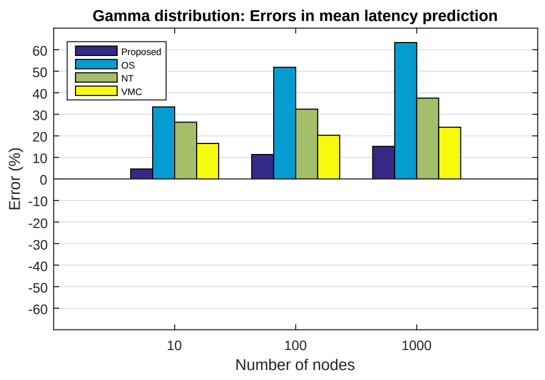
3.4.2 Scenario 2: M/G/1 Queues

In this case study, we consider non-exponential service time distribution for each queue. In particular, we evaluate the accuracy of the proposed approach against the direct approximation based on the order statistic for Gamma, Weibull, truncated Pareto (with $CV = 1.2$, which is not in the lookup table), and empirical distributions. We also include the approximations from Eqs. (3.1) and (3.3), which are tailored to the M/M/1 queues, just for references to show that they are worse when applied to the queues with non-exponential service time distributions.

The comparative results are shown in Fig. 3.8–3.11. From all those figures, one can see that the approximations in Eqs. (3.1) and (3.3) cannot be applied to general service time distributions. For truncated Pareto and empirical distributions, i.e., heavy-tailed, in some cases, e.g., at the load of 80%, the direct approximation from order statistic yields better accuracy than the proposed approach with ratio offset using the 99th percentile and mean latency ratios. However, at the load of 90%, where the tail latency prediction is more accurate, the proposed approach always gives better results. For less heavy-tailed distributions, i.e, Gamma and Weibull, the proposed approximation yields the best results for all the cases under study. Fortunately, the tail effect is a well-aware issue in datacenter applications, so tail-cutting techniques are often utilized in datacenters to curb the tail effects [20, 29, 33, 54, 61]. As a result, the effective service time distributions tend to be less heavy-tailed. Therefore, the proposed approach shows a great potential to be applied to the approximation of mean latency in datacenter applications, especially for resource provisioning and scaling performance study, which are of our interest.

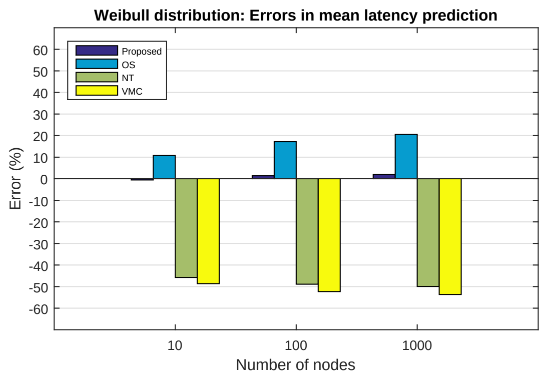


(a) Load of 80%

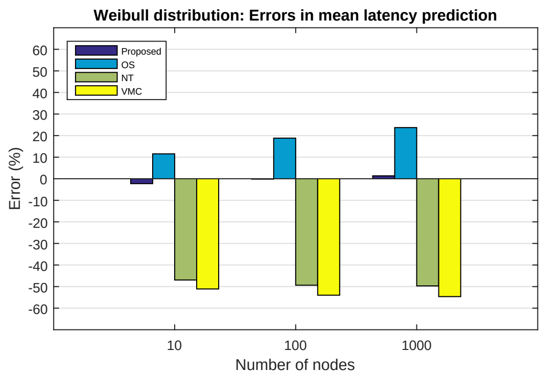


(b) Load of 90%

Figure 3.8: Gamma distribution: Comparison of mean response time approximation techniques.

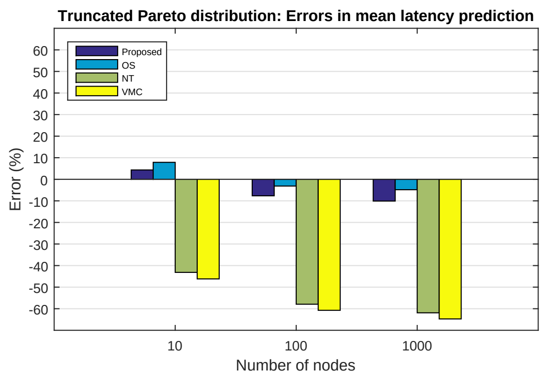


(a) Load of 80%

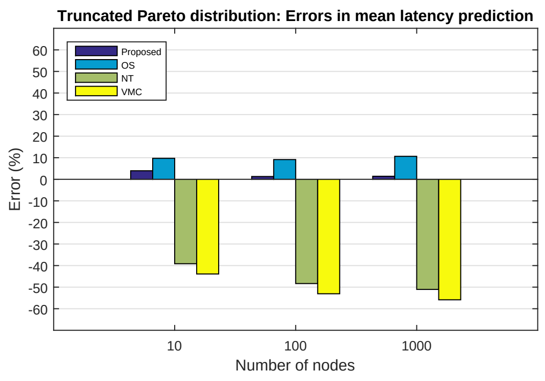


(b) Load of 90%

Figure 3.9: Weibull distribution: Comparison of mean response time approximation techniques.

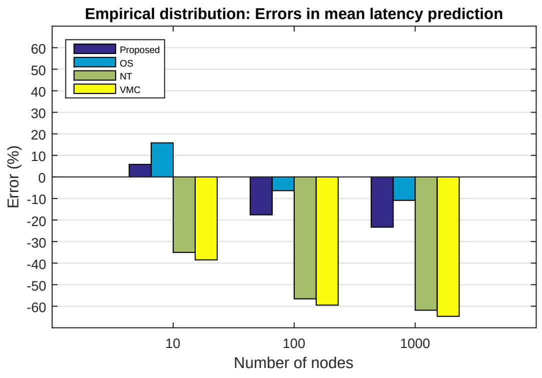


(a) Load of 80%

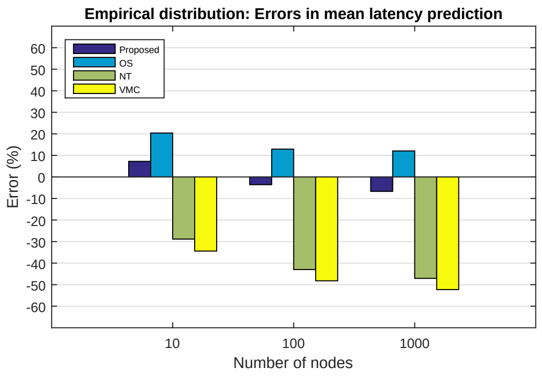


(b) Load of 90%

Figure 3.10: Truncated Pareto distribution: Comparison of mean response time approximation techniques.



(a) Load of 80%



(b) Load of 90%

Figure 3.11: Empirical distribution: Comparison of mean response time approximation techniques.

3.5 Conclusions

In this chapter, we proposed the analytical approaches for tail and mean latency prediction. The tail latency prediction model is an alternate to the approach in chapter 2. It models each subsystem in the system as an M/G/1 queue and provides an analytical solution for the prediction. This approach requires a known service time distribution whose first three moments exist or empirical data from a test subsystem for the calculation of the required sample moments. The approach is highly suitable for offline resource provisioning as it is independent of job flow traffic patterns. The mean latency prediction model is an empirical approach based on the predicted tail latency. It also provides an accurate prediction for mean latency when the system is under heavy traffic, especially for less heavy-tailed service time distributions. This shows a great potential to be applied to datacenter applications, which usually utilize tail-cutting techniques to reduce the tail effect.

Chapter 4

PERFORMANCE SCALING ANALYSIS OF SCALE-OUT APPLICATIONS

4.1 Introduction

Today's datacenter applications, such as search engine, social networking, data serving, and media streaming, have to process big data sets and provide fast responses to user requests. Therefore, they are scale out by design based on programming models, e.g., MapReduce [21], Spark [2], or Dryad [31], to utilize resources at the scale of datacenters. Having good understanding of scaling properties of such applications thus help users as well as datacenter providers make use of resources more efficiently.

Unfortunately, traditional scaling laws for parallel computing, Amdahl's law [7] and its variants, e.g., Gustafson's law [28] and Sun and Ni's law [53], are no longer adequate to characterize the scaling properties of datacenter applications. They primarily focus on the workload space exploration. For instance, Amdahl's law [7] captures the limitation of performance scaling for fixed-size workload. It states that with given percentage of parallelizable part of a fixed-size program, η, even with unlimited number of parallel processing units, the speedup is fundamentally limited by $1/(1-\eta)$. On the contrary, Gustafson's law [28] characterizes performance scaling for fixed-time workloads. That is, assuming that the parallelizable part of the program grows linearly with the number of parallel processing units, the speedup grows unboundedly with the increasing number of parallel processing units. The rationale behind Gustafson's law is the observation that in practice, the problem size can be expanded to make use of the increased resources. Sun and Ni's law [53] on memory-bounded speedup again predicts unbounded speedup, which applies to the problem whose size scales to the total memory capacity. All of these laws consider only the scaling of the parallelizable portion of the workload, which is called *external scaling* in this chapter, and assume the serial portion unchanged. This assumption is inadequate for the scaling analysis of scale-out applications since the serial portion, which corresponds to the merging of partial results from the parallel tasks, may scale with the number of tasks or processing units, which is referred to as *internal scaling*.

Also, the existing scaling laws did not consider overhead due to scaling out workload, which is usually the case in scale-out applications due to task scheduling [30, 49, 60], data block dispatching or update [19]. These overheads significantly limit the scalability of distributed programming frameworks, such as Hadoop [1] or Spark [2], which slow down the speedup or even make it "negative" if the overhead dominates the speedup gain from workload parallelization. Several relevant works in the context of parallel computing have pointed out the effective serial workload introduced by external scaling due to reduction operation [41] or critical sections [23]. In [16], we generalized these effects using a queuing-network-based analysis and showed that any resource access contention, whether it is a shared resource or critical section, is guaranteed to induce an effective serial workload.

Moreover, the scaling models in the existing laws consider parallel tasks of the workload as deterministic ones, assuming they are completed at the same time. In other words, they did not take probabilistic nature, i.e., variations in task completion times, into account. Datacenter applications usually need to be processed by multiple stages where the next stage may not start until all the tasks from the previous stage finish. As a result, the job response time at a given stage is determined by its slowest task, which is known as *barrier synchronization*. A relevant work [64] showed that the accuracy of the speedup evaluation is improved when incorporating this synchronization effect.

The above scaling factors were modeled without queuing effect and tested in [39]. It showed that the proposed model is able to capture the scaling properties of various real scale-out applications. However, this model only applies to workloads with a single job. In practice, jobs or requests may be sent to a system for processing in a continuous flow with average arrival rate λ. Therefore, jobs and their tasks need to be queued at corresponding nodes in the system. The work in this chapter generalizes the model in [39] to incorporate all the above scaling factors, i.e., the external scaling, internal scaling, and barrier synchronization as well as the queuing effect at each processing node in the system. With the approximate solution developed in chapter 3, we are able to derive a closed-form solution for this model. This solution becomes more accurate as the load increases for heavy-tailed task service time distributions and is accurate in the entire load range for light-tailed task service time distributions. A case study is also given for the case where all the task service times are exponentially distributed, i.e., light-tailed.

The remainder of this chapter is organized as follows. Section 4.2 proposes the workload model for the scaling analysis. Section 4.3 presents an analytical model for a case study with exponential service time distributions. Finally, Section 4.4 concludes the chapter and discusses the future work.

4.2 The Workload Model

In this chapter, we consider a system modeling scale-out [21] applications as in Fig. 4.1. The map stage is modeled as a n-node Fork-Join queuing network as in chapter 3 in which each processing node can generally be modeled as a -/G/1 queuing model, i.e., an arbitrary arrival process and general service time distribution with one service center. A job coming to the system with an arrival rate of λ is forked into n map tasks. All the tasks belonging to a job need to be completed in the map stage before the job can be sent to the reduce stage, a.k.a *barrier synchronization*. We assume $D_{p,i}(n)$ and $T_{p,i}(n)$ are random variables corresponding to the response time and service time of map task i, respectively. The reduce stage, which merges partial results from the map stage, is modeled as a -/G/1 queuing system with service time $T_s(n)$, i.e., an arbitrary arrival process, general service times with mean $\mathbb{E}[T_s(n)]$, and one service center. The additional queue, i.e., the first stage, represents the effective workload corresponding to the scale-out overhead. It is modeled as an M/M/1 queuing system with mean service time $\mathbb{E}[T_o(n)]$ being proportional to the incurred overhead. We assume the arrival of jobs at this node is a Poisson process, which was found to be adequate for scale-out applications [44].

4.2.1 Without Queuing

Here we consider a special case when there is no queuing at each node in the system, i.e., $D_{p,i}(n) = T_{p,i}(n)\ \forall i$ in Fig. 4.1. This workload model and its scaling properties were

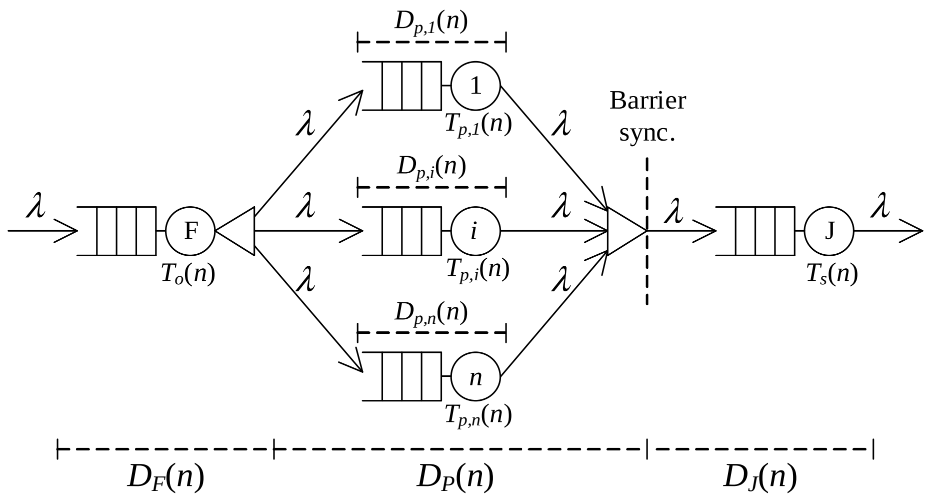


Figure 4.1: A system model with a scale-out-induced stage (F), a map stage (P), and a reduce stage (J). All the tasks belonging to a job need to be completed in the map stage before the job can be sent to the reduce stage.

proposed and studied in [39], which are summarized in the following.

Assume that the map portion are homogeneous, i.e., all the map tasks have the same mean processing time, $\mathbb{E}[T_{p,i}(n)] = \mathbb{E}[T_p(n)]\ \forall i$. The total procesing time for the parallelizable portion of the workload,

$$W_p(n) = \mathbb{E}\left[\sum_{i=1}^{n} T_{p,i}(n)\right] = n\mathbb{E}[T_p(n)] = \mathbb{E}[T_p(1)]h(n)$$

Therefore, we have,

$$\mathbb{E}[T_p(n)] = \frac{\mathbb{E}[T_p(1)]h(n)}{n}, \tag{4.1}$$

where $\mathbb{E}[T_p(1)]$ is the mean processing time for the parallelizable portion of the workload when the system has only one processing node and $h(n)$ is a scaling factor, which is called *external scaling factor* [39]. In general, this scaling factor is linearly proportional to the number of parallel nodes n.

Similarly, the total procesing time for the serial portion of the workload,

$$W_s(n) = \mathbb{E}[T_s(n)] = \mathbb{E}[T_s(1)]v(n), \tag{4.2}$$

where $\mathbb{E}[T_s(1)]$ is the mean processing time for the serial portion of the workload when the system has only one processing node and $v(n)$ is a in-proportion scaling factor or *internal scaling factor* [39]. This scaling factor is generally proportional to n due to the merging process of partial results from the parallel stage.

The *in-proportion scaling ratio* is defined as,

$$\epsilon(n) = \frac{h(n)}{v(n)}\,. \tag{4.3}$$

When n becomes large, one can approximate $\epsilon(n)$ as follows,

$$\epsilon(n) = \alpha n^{\delta}, \tag{4.4}$$

where $\alpha > 0$. In general, one can assume $\delta \leq 1$ since $h(n)$ is not scaled more than a linear order of n and $v(n)$ is not scaled down.

Moreover, when the system is scaled out, it may be induced an overhead due to task scheduling [30, 60], data-block dispatching [49], data broadcast or parameter update [19]. This introduces an effective workload, $W_o(n)$, to the overall workload, which is called *scale-out-induced workload*. It should be proportional to the block size of the parallelizable workload and is defined as

$$W_o(n) = \mathbb{E}[T_o(n)] = \frac{W_p(n)}{n}q(n) = \frac{\mathbb{E}[T_p(1)]h(n)}{n}q(n)\,, \tag{4.5}$$

where $q(n)$ is called *scale-out-induced scaling factor*, which is a non-decreasing function of n and equal to zero when $n = 1$.

Finally, due to the barrier synchronization and the randomness of task processing time, the mean job response time is determined by the slowest task processing time, i.e., $\mathbb{E}[\max\{T_{p,i}(n)\}]$.

The speedup with respect to processing time can be define as

$$\begin{aligned} S_d(n) &= \frac{W_p(n) + W_s(n)}{\mathbb{E}[\max\{T_{p,i}(n)\}] + W_s(n) + W_o(n)} \\ &= \frac{\mathbb{E}[T_p(1)]h(n) + \mathbb{E}[T_s(1)]v(n)}{\mathbb{E}[\max\{T_{p,i}(n)\}] + \mathbb{E}[T_s(1)]v(n) + \frac{\mathbb{E}[T_p(1)]h(n)}{n}q(n)} \\ &= \frac{\eta h(n) + (1-\eta)v(n)}{\frac{\mathbb{E}[\max\{T_{p,i}(n)\}]}{\mathbb{E}[T_p(1)] + \mathbb{E}[T_s(1)]} + (1-\eta)v(n) + \frac{\eta h(n)}{n}q(n)}, \end{aligned} \tag{4.6}$$

where η is the percentage of the parallelizable portion of the job workload at $n = 1$, i.e.,

$$\eta = \frac{W_p(1)}{W_p(1) + W_s(1)} \equiv \frac{\mathbb{E}[T_p(1)]}{\mathbb{E}[T_p(1)] + \mathbb{E}[T_s(1)]} . \tag{4.7}$$

Consider a special case when all the parallel tasks are homogeneous and deterministic, i.e.,

$$\mathbb{E}[\max\{T_{p,i}(n)\}] = \mathbb{E}[T_p(n)] = \frac{\mathbb{E}[T_p(1)]h(n)}{n} . \tag{4.8}$$

Substituting Eqs. (4.8) and (4.7) into Eq. (4.6), we have,

$$\begin{aligned} S_d(n) &= \frac{\eta h(n) + (1-\eta)v(n)}{\frac{\eta h(n)}{n} + (1-\eta)v(n) + \frac{\eta h(n)}{n}q(n)} \\ &= \frac{\eta h(n) + (1-\eta)v(n)}{\frac{\eta h(n)}{n}(1 + q(n)) + (1-\eta)v(n)} . \end{aligned} \tag{4.9}$$

Let $v(n) = 1$ and $q(n) = 0$, i.e., without considering the possible in-proportion and scale-out-induced scaling, and,

$$h(n) = \begin{cases} 1, & \text{for fixed-size workload,} \\ n, & \text{for fixed-time workload,} \\ \overline{g}(n), & \text{for memory-bounded workload,} \end{cases}$$

where $\overline{g}(n)$ is a scaling factor that is constrained by the total memory size.

The well-known scaling laws can be derived from the speedup in Eq. (4.9),

$$S_d(n) = \begin{cases} \dfrac{1}{\dfrac{\eta}{n} + (1-\eta)}, & \text{Amdahl's law,} \\ \eta n + (1-\eta), & \text{Gustafson's law,} \\ \dfrac{\eta\overline{g}(n) + (1-\eta)}{\dfrac{\eta\overline{g}(n)}{n} + (1-\eta)}, & \text{Sun-Ni's law.} \end{cases} \tag{4.10}$$

4.2.2 With Queuing

We now consider a case in which a flow of jobs comes into the system with average arrival rate λ as in Fig. 4.1. Each node in the system thus needs a queue for jobs/tasks waiting for processing. All the queuing disciplines are assumed to be First-Come-First-Served (FCFS). In the following, we summarize the BCMP queuing networks [13] and its solution for an open queuing network that is used to approximate the scale-out model for scaling analysis later.

4.2.2.1 BCMP Queuing Networks

It was proven that for a large class of queuing networks known as the BCMP queuing networks, closed-form solutions exist. This class of queuing networks permits arbitrary network topologies, i.e., open, closed, and mixed networks, several job classes, different queuing disciplines, and general service time distributions [13]. The following four types of queuing nodes fulfill conditions for the BCMP networks:

– Type 1: -/M/m/FCFS (First-Come-First-Served),

– Type 2: -/G/1/PS (Processor Sharing),

– Type 3: -/G/∞ (IS - Infinite Server),

– Type 4: -/G/1/LCFS-PR (Last-Come-First-Served - Preemptive Resume).

For all four types, the hyphen symbols at the beginning simply mean that in a BCMP queuing network, the arrival process to any queuing node may not be Poisson. Type 1 nodes have exponential service times, denoted as M, with m servers and First-Come-First-Served queuing discipline. Symbol G in Types 2–4 means that the service time may follow any distribution function. Type 3 nodes have infinite number of servers (IS) equivalent to delay nodes. Type 2 and 4 nodes have Processor Sharing (PS) and Last-Come-First-Served Preemptive Resume (LCFS-PR) queuing disciplines, respectively.

Consider an open queuing network with a single job class in Fig. 4.2. All the nodes in this network satisfy the assumptions of the BCMP networks: -/M/1/FCFS for node 1 (F), -/G/∞ for node 2 (P), and -/G/∞ for node 3 (J).

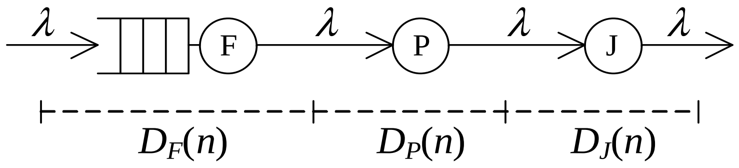


Figure 4.2: An equivalent queuing model for the system model in Fig. 4.1.

According to BCMP theorem [13], with a single job class, the *mean number of jobs* at node i, for $i = 1, 2, 3$, is given by

$$\overline{K_i} = \frac{\rho_i}{1 - \rho_i}\,, \tag{4.11}$$

where ρ_i is server utilization at node i,

$$\rho_i = \frac{\lambda}{\mu_i} \, , \tag{4.12}$$

where λ is the *mean arrival rate* to the network and μ_i is the *mean service rate* of node i.

The *overall mean response time* of the network can be derived using Little's law as follows,

$$\mathbb{E}[D(n)] = \frac{\overline{K}}{\lambda} = \frac{1}{\lambda}\sum_{i=1}^{3}\overline{K}_i = \frac{1}{\lambda}\sum_{i=1}^{3}\frac{\rho_i}{1-\rho_i} \, . \tag{4.13}$$

4.2.2.2 Sequential Model

For the speedup calculation, we need a sequential model as a reference. To build the model, we assume that there is only one processing unit to handle the entire workload. This is equivalent to the fact that all the tasks in the parallelizable portion are flattened out along with the serial portion as described in Fig. 4.3.

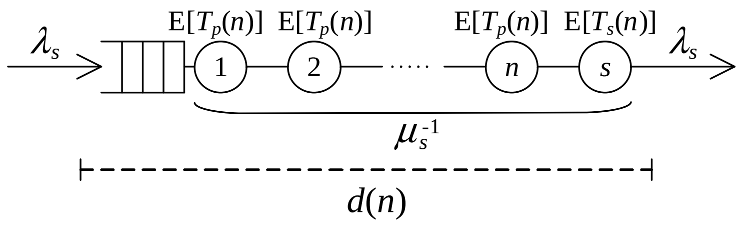


Figure 4.3: A proposed sequential queuing model with its service center is a pipeline of n parallelizable stages and a sequential stage.

One can view the service time for this queuing model as a sum of service times of individual tasks. According to Central Limit Theorem [57], as n becomes large, the service time distribution can be approximated as a Normal distribution with mean and variance as follows,

$$\begin{aligned} \mu_s^{-1} &= n\mathbb{E}[T_p(n)] + \mathbb{E}[T_s(n)]\,, && (4.14) \\ \sigma_s^2 &= n\sigma_P^2 + \sigma_J^2\,, && (4.15) \end{aligned}$$

where μ_s is the mean service rate, σ_P^2 and σ_J^2 are variances of service times of the map and reduce nodes, respectively.

Here we also assume that the arrival process is Poisson, the model thus can be viewed as an M/G/1 queuing model with mean response time being given by [13]

$$\mathbb{E}[d(n)] = \mu_s^{-1}\left(1 + \frac{\rho_s}{1-\rho_s} \cdot \frac{1+C_s^2}{2}\right) = f(\lambda_s; \mu_s^{-1}, \sigma_s), \tag{4.16}$$

which is a function of the arrival rate or throughput λ_s and parameterized by μ_s^{-1} and σ_s, where $\rho_s = \lambda_s\mu_s^{-1}$ is server utilization and $C_s^2 = \sigma_s^2/(\mu_s^{-1})^2$ is coefficient of variation of service times.

4.2.2.3 Scale-out Model

We observe that the overall mean response time in Eq. (4.13) of the open queuing network above can be viewed as a sum of mean response times from all the stages, i.e.,

$$\mathbb{E}[D(n)] = \mathbb{E}[D_F(n)] + \mathbb{E}[D_P(n)] + \mathbb{E}[D_J(n)] \tag{4.17}$$

This observation suggest that the workload model in Fig. 4.1 can be simplified and approximated by the open queuing network in Fig. 4.2 as explained below.

The first stage: M/M/1 queuing model

With the assumption that the arrival of jobs at the system is a Poisson process, the first stage F in Fig. 4.1, representing the overhead due to scaling out the workload can be modeled as an M/M/1 queuing model with its mean service time, i.e., $\mathbb{E}[T_o(n)]$, being proportional to the incurred overhead. This correctly matches stage F in Fig. 4.2.

The mean response time of this stage is given as

$$\mathbb{E}[D_F(n)] = \frac{\mu_F^{-1}}{1-\rho_F} , \tag{4.18}$$

where μ_F^{-1} is the mean service time,

$$\mu_F^{-1} = \mathbb{E}[T_o(n)] = \frac{\mathbb{E}[T_p(1)]h(n)}{n} q(n) . \tag{4.19}$$

The second stage: -/G/∞ queuing model

Since scale-out-induced stage F is an M/M/1 queuing model, according to Burke's theorem [57], its departure process is a Poisson process. Therefore, the arrival process at the second stage is also a Poisson process. Accordingly, each node in this stage can be considered an M/G/1 queuing system, i.e., Poisson arrival process with average rate λ and general service times. This queuing network is equivalent to the Fork-Join queuing network considered in chapter 3. We thus can utilize the approximation technique developed in chapter 3 to find the mean response time of this stage. As a result, the whole map stage in Fig. 4.1 can be simplified as a delay node, which can be approximated by a -/G/∞ queuing model as stage P in Fig. 4.2. Note that the mean response time of this stage is calculated at given average arrival rate λ. In other words, it is a function of λ.

Assume that the response time CDF of each node in this stage is approximated by a generalized exponential distribution at given λ,

$$F_X^P(x) = (1 - e^{-x/\beta_1})^{\alpha_1} , \tag{4.20}$$

where α_1 and β_1 are shape and scale parameter, respectively. These two parameters can be computed given the first three moments of $T_{p,i}(n)$.

The mean response time of this stage can be approximated using the approach proposed in chapter 3,

$$\mathbb{E}[D_P(n)] \approx \frac{x_p}{R_P^{ge}(n) + \Delta} \, , \tag{4.21}$$

where Δ is the 99th/mean ratio difference between the experiment result and the prediction model, which is looked up from a pre-built table based on the server utilization at each node and the tail weight of service time distribution, and

$$\begin{aligned} x_p &\approx x_p^{ge} = -\beta_1 \log\left(1 - q^{\frac{1}{n\alpha_1}}\right) \quad \text{with } q = p/100 \, , &(4.22) \\ R_P^{ge}(n) &= \frac{x_p^{ge}}{\mathbb{E}[D_P^{ge}(n)]} = \frac{-\log\left(1 - q^{\frac{1}{n\alpha}}\right)}{\psi(n\alpha_1 + 1) - \psi(1)} \, . &(4.23) \end{aligned}$$

The third stage: -/G/∞ queuing model

The queuing model of stage J is similar to that of a node in stage P, which is a -/G/1 with the mean service time given as,

$$\mu_J^{-1} = \mathbb{E}[T_s(n)] = \mathbb{E}[T_s(1)]v(n) \, . \tag{4.24}$$

In general, the arrival process of this stage, which is the departure process from the second stage, is not a Poisson one. However, here it is approximated as a Poisson process, i.e., becoming an M/G/1 queuing model. As one shall see from the extensive simulation-based verification, the approximation is adequate to be used for performance scaling analysis.

We again approximate the response time distribution of this queue, i.e., M/G/1, using the generalized exponential distribution as in chapter 3, assuming the first three moments of $T_s(n)$ exist,

$$F_X^J(x) = (1 - e^{-x/\beta_2})^{\alpha_2} \, , \tag{4.25}$$

Given this distribution, one can find its mean response time,

$$\mathbb{E}[D_J(n)] = \beta_2[\psi(\alpha_2 + 1) - \psi(1)] \tag{4.26}$$

A special case when service time distribution G is exponential, we have,

$$\begin{aligned} F_X^J(x) &= 1 - e^{-\mu_J(1-\rho_J)x}, \\ \mathbb{E}[D_J(n)] &= \frac{\mu_J^{-1}}{1-\rho_J}. \end{aligned}$$

With this approximation, similar to the map stage, this stage can be approximated as a delay node, i.e., a -/G/∞ queuing model, where G is $F_X^J(x)$.

4.2.3 Simulation-based Verification of the Proposed Approximation

In this section, we evaluate the accuracy of the proposed approximation model through simulation with different experimental settings. We run simulations for the system in Fig. 4.1 and compare the overall response times against the approximation using Eq. (4.17). For simplicity, we assume both scale-out-induced and reduce stages are M/M/1 queuing models. We run simulations with different service time distributions for the map stage's nodes, including

– Exponential distribution: $CV = 1.0$,

– Weibull distribution: $CV = 1.5$,

– Truncated Pareto distribution: $CV = 2.0$,

– Empirical distribution: $CV = 1.12$.

Tables. 4.1–4.4 show the relative errors of the proposed approximation versus the simulated values. For all the cases, the mean service time of each node in the map stage is considered to be one. The mean service times of scale-out-induced and reduce stages are presented as factors relative to the one from the map stage. As one can see from these tables, most cases have errors within 10%, especially at high loads since this mean latency approximation is based on the tail latency prediction model which is more accurate at high loads. This approximation model is adequate for performance scaling analysis, which is studied in the next section.

Table 4.1: Relative errors of the proposed approximation for exponential distribution.

		Mean service time						
		$\mu_F = 1.0$	$\mu_F = 2.0$	$\mu_F = 1.0$	$\mu_F = 2.0$	$\mu_F = 0.8$	$\mu_F = 1.0$	$\mu_F = 0.8$
#nodes	**Load**	$\mu_J = 1.0$	$\mu_J = 1.0$	$\mu_J = 2.0$	$\mu_J = 2.0$	$\mu_J = 1.0$	$\mu_J = 0.8$	$\mu_J = 0.8$
100	50%	2.899	3.262	1.961	2.111	1.964	2.506	2.440
	75%	4.418	4.810	3.274	3.182	3.055	4.319	4.050
	80%	4.701	5.310	3.003	3.857	3.212	4.647	4.149
	90%	5.367	5.625	2.466	3.684	3.077	4.466	3.389
1000	50%	4.209	4.672	3.313	3.666	2.877	3.608	3.462
	75%	6.335	6.689	5.002	5.537	4.686	5.675	5.423
	80%	6.641	7.190	5.119	5.786	4.326	6.269	5.582
	90%	6.264	7.059	5.229	5.993	3.757	6.924	5.175

Table 4.2: Relative errors of the proposed approximation for Weibull distribution.

		Mean service time						
		$\mu_F = 1.0$	$\mu_F = 2.0$	$\mu_F = 1.0$	$\mu_F = 2.0$	$\mu_F = 0.8$	$\mu_F = 1.0$	$\mu_F = 0.8$
#nodes	**Load**	$\mu_J = 1.0$	$\mu_J = 1.0$	$\mu_J = 2.0$	$\mu_J = 2.0$	$\mu_J = 1.0$	$\mu_J = 0.8$	$\mu_J = 0.8$
100	50%	-3.154	-3.333	-1.939	-2.222	-3.919	-4.664	-4.362
	75%	-1.132	-1.122	1.318	1.128	-2.069	-3.271	-3.232
	80%	-0.762	-0.493	1.550	1.733	-1.558	-3.019	-2.957
	90%	-0.477	-1.223	2.057	2.623	-0.961	-2.739	-2.358
1000	50%	-5.179	-5.472	-4.357	-4.687	-7.236	-7.770	-7.616
	75%	-0.010	-0.399	1.786	1.571	-2.671	-3.624	-3.301
	80%	0.689	0.480	2.613	2.653	-1.846	-2.832	-2.860
	90%	1.254	1.326	2.912	3.254	-0.212	-1.348	-0.618

Table 4.3: Relative errors of the proposed approximation for truncated Pareto distribution.

		Mean service time						
#nodes	**Load**	$\mu_F = 1.0$ $\mu_J = 1.0$	$\mu_F = 2.0$ $\mu_J = 1.0$	$\mu_F = 1.0$ $\mu_J = 2.0$	$\mu_F = 2.0$ $\mu_J = 2.0$	$\mu_F = 0.8$ $\mu_J = 1.0$	$\mu_F = 1.0$ $\mu_J = 0.8$	$\mu_F = 0.8$ $\mu_J = 0.8$
100	50%	-10.016	-10.217	-5.909	-6.066	-9.847	-13.413	-13.192
	75%	-5.599	-5.805	-0.596	-0.487	-8.409	-14.476	-14.034
	80%	-4.169	-4.691	0.886	1.331	-7.760	-14.433	-13.932
	90%	-1.133	-2.541	4.867	4.586	-5.741	-15.213	-13.701
1000	50%	-8.089	-8.412	-6.225	-6.403	-12.897	-14.201	-14.115
	75%	-0.940	-0.780	2.672	2.961	-4.301	-7.929	-7.578
	80%	-0.257	-0.159	3.809	4.223	-3.316	-7.436	-7.305
	90%	1.630	2.010	6.254	7.418	-1.269	-8.495	-8.241

Table 4.4: Relative errors of the proposed approximation for the empirical distribution.

		Mean service time						
#nodes	**Load**	$\mu_F = 1.0$ $\mu_J = 1.0$	$\mu_F = 2.0$ $\mu_J = 1.0$	$\mu_F = 1.0$ $\mu_J = 2.0$	$\mu_F = 2.0$ $\mu_J = 2.0$	$\mu_F = 0.8$ $\mu_J = 1.0$	$\mu_F = 1.0$ $\mu_J = 0.8$	$\mu_F = 0.8$ $\mu_J = 0.8$
100	50%	-17.762	-18.764	-14.333	-15.356	-14.855	-18.501	-17.833
	75%	-16.816	-17.725	-12.706	-14.056	-17.527	-24.421	-22.942
	80%	-13.934	-14.809	-10.371	-11.273	-17.415	-25.038	-22.970
	90%	-0.761	-0.611	2.573	3.032	-15.173	-23.722	-20.488
1000	50%	-37.472	-38.371	-36.848	-37.858	-40.467	-41.769	-41.088
	75%	-21.540	-22.446	-20.782	-21.615	-31.213	-33.301	-32.055
	80%	-16.284	-17.221	-15.215	-16.110	-28.517	-30.793	-29.417
	90%	-0.972	-1.266	1.987	2.274	-21.854	-24.861	-22.640

4.2.4 Speedup

The response time speedup can be defined as a ratio between the response time of the sequential model and that of the scale-out model at a given arrival rate λ_0 for both models, i.e.,

$$\begin{aligned} S_d(n) &= \left.\frac{\mathbb{E}[d(n)]}{\mathbb{E}[D(n)]}\right|_{\lambda=\lambda_s=\lambda_0} \\ &= \left.\frac{\mathbb{E}[d(n)]}{\mathbb{E}[D_F(n)] + \mathbb{E}[D_P(n)] + \mathbb{E}[D_J(n)]}\right|_{\lambda=\lambda_s=\lambda_0}, \end{aligned} \tag{4.27}$$

where $\mathbb{E}[d(n)]$, $\mathbb{E}[D_F(n)]$, $\mathbb{E}[D_P(n)]$, and $\mathbb{E}[D_J(n)]$ are given in Eqs. (4.16), (4.18), (4.21), and (4.26), respectively. Note that for the stability of queuing model, the arrival rate is limited by the service rate, i.e., the utilization ρ_0 must be less than one ($0 \leq \rho_0 < 1$). Also, the arrival rate for the sequential model is usually much less than that for the scale-out model. Therefore, the response time speedup in Eq. (4.27) is calculated with λ_0 ranging from zero to maximum possible value for the sequential model $\lambda_0^{\max}$ ($\lambda_0^{\max} < \mu_s$). The response time for the sequential model is not defined beyond $\lambda_0^{\max}$. It means that, to achieve a system throughput beyond the service rate limit of the sequential model, scaling out the workload is the only way to go.

The throughput speedup is defined as a ratio of the throughput of the scale-out model and that of the sequential model at a given response time or latency constraint d_0 ,

$$S_\lambda(n) = \left.\frac{\lambda}{\lambda_s}\right|_{d=D=d_0} = \left.\frac{g^{-1}(\mathbb{E}[D(n)])}{f^{-1}(\mathbb{E}[d(n)])}\right|_{d=D=d_0}, \tag{4.28}$$

where $f^{-1}(.)$ and $g^{-1}(.)$ are the inverse functions of the mean response times of the sequential and scale-out models, respectively. Since the mean response time of the sequential model is usually larger than that of the scale-out model, we use it as a reference in the throughput speedup calculation. Note that its value is at least the mean service time, i.e., $d_0 \geq \mu_s^{-1}$.

4.3 Case Study: Exponential Service Time Distribution

Consider the case in which all the service times at the queuing nodes are exponentially distributed.

The latency speedup is calculated at the same arrival rate, λ_0, for both sequential and scale-out model,

$$S_d(n) = \left.\frac{\mathbb{E}[d(n)]}{\mathbb{E}[D(n)]}\right|_{\lambda=\lambda_s=\lambda_0}$$
$$= \left.\frac{\mathbb{E}[d(n)]}{\mathbb{E}[D_F(n)] + \mathbb{E}[D_P(n)] + \mathbb{E}[D_J(n)]}\right|_{\lambda=\lambda_S=\lambda_0} \tag{4.29}$$

The sequential model

From Eq. (4.14) and (4.16), the response time for the sequential model is given by

$$\mathbb{E}[d(n)] = \mu_s^{-1}\left(1 + \frac{\rho_0}{1-\rho_0} \cdot \frac{1+C_s^2}{2}\right)$$
$$= (n\mathbb{E}[T_p(n)] + \mathbb{E}[T_s(n)])\left(1 + \frac{\rho_0}{1-\rho_0} \cdot \frac{1+C_s^2}{2}\right) , \tag{4.30}$$

where $\rho_0 = \lambda_0 \mu_s^{-1}$ is the server utilization ($0 \leq \rho_0 < 1$).

The corresponding arrival rate λ_0 can be written as,

$$\lambda_0 = \frac{\rho_0}{\mu_s^{-1}} = \frac{\rho_0}{n\mathbb{E}[T_p(n)] + \mathbb{E}[T_s(n)]} , \tag{4.31}$$

which is a function of n and depends on the workload type under study, e.g., fixed-size or fixed-time workload. In what follows, we use this arrival rate for the calculation of the response time of the scale-out model as well as the response time speedup.

The scale-out model

From Eqs. (4.17), (4.18), (4.19), (4.21), (4.24), (4.26), and (4.31), we can derive the following relations,

$$\mathbb{E}[D(n)] = \mathbb{E}[D_F(n)] + \mathbb{E}[D_P(n)] + \mathbb{E}[D_J(n)] \tag{4.32}$$

$$\begin{aligned}\mathbb{E}[D_F(n)] &= \frac{\mu_F^{-1}}{1-\rho_F} = \frac{\mu_F^{-1}}{1-\lambda_0\mu_F^{-1}} = \frac{\mathbb{E}[T_p(n)]q(n)}{1-\dfrac{\rho_0\mathbb{E}[T_p(n)]q(n)}{n\mathbb{E}[T_p(n)]+\mathbb{E}[T_s(n)]}} \\ &= \frac{\mathbb{E}[T_p(n)]q(n)\cdot(n\mathbb{E}[T_p(n)]+\mathbb{E}[T_s(n)])}{n\mathbb{E}[T_p(n)]\left(1-\rho_0\dfrac{q(n)}{n}\right)+\mathbb{E}[T_s(n)]} \end{aligned} \tag{4.33}$$

$$\begin{aligned}\mathbb{E}[D_J(n)] &= \frac{\mu_J^{-1}}{1-\rho_J} = \frac{\mu_J^{-1}}{1-\lambda_0\mu_J^{-1}} = \frac{\mathbb{E}[T_s(n)]}{1-\dfrac{\rho_0\mathbb{E}[T_s(n)]}{n\mathbb{E}[T_p(n)]+\mathbb{E}[T_s(n)]}} \\ &= \frac{\mathbb{E}[T_s(n)]\cdot(n\mathbb{E}[T_p(n)]+\mathbb{E}[T_s(n)])}{n\mathbb{E}[T_p(n)]+\mathbb{E}[T_s(n)](1-\rho_0)} \end{aligned} \tag{4.34}$$

$$\mathbb{E}[D_P(n)] = \frac{x_p}{R_P^{ge}(n)+\Delta}$$

which can be calculated using the proposed approximation method in chapter 3,

$$\begin{aligned} x_p \approx x_p^{ge} &= -\frac{1}{\mu_P-\lambda_0}\log\left(1-q^{1/n}\right) \quad \text{with } q = p/100\,, \\ \mathbb{E}[D_P^{ge}(n)] &= \frac{1}{\mu_P-\lambda_0}\mathcal{H}_n\,, \\ R_P^{ge}(n) &= \frac{x_p^{ge}}{\mathbb{E}[D_P^{ge}(n)]} = \frac{-\log(1-q^{1/n})}{\mathcal{H}_n}\,, \\ \mathbb{E}[D_P(n)] &= \frac{x_p}{R_P^{ge}(n)+\Delta} = \frac{\mu_P^{-1}\cdot C_{par}}{1-\lambda_0\mu_P^{-1}} = \frac{\mathbb{E}[T_p(n)]\cdot C_{par}}{1-\dfrac{\rho_0\mathbb{E}[T_p(n)]}{n\mathbb{E}[T_p(n)]+\mathbb{E}[T_s(n)]}} \\ &= \frac{\mathbb{E}[T_p(n)]\cdot C_{par}\cdot(n\mathbb{E}[T_p(n)]+\mathbb{E}[T_s(n)])}{n\mathbb{E}[T_p(n)]\left(1-\dfrac{\rho_0}{n}\right)+\mathbb{E}[T_s(n)]} \end{aligned} \tag{4.35}$$

where

$$C_{par} = \frac{-\log\left(1-q^{1/n}\right)}{\dfrac{-\log\left(1-q^{1/n}\right)}{\mathcal{H}_n}+\Delta}\,, \tag{4.36}$$

where Δ is the 99th/mean ratio difference between the experiment result and the prediction model which is looked up from a pre-built table and $\mathcal{H}_n$ is the nth harmonic number, $\mathcal{H}_n =$

$\sum_{k=1}^{n} \frac{1}{k}$. Note that when n becomes large, C_{par} is proportional to $\mathcal{H}_n$, which slowly increases with n.

Substituting Eqs. (4.30)–(4.36) into (4.29), we have,

$$\begin{aligned} S_d(n) = &\left(1 + \frac{\rho_0}{1-\rho_0} \cdot \frac{1+C_s^2}{2}\right) \Bigg/ \\ &\left(\frac{\mathbb{E}[T_p(n)]q(n)}{n\mathbb{E}[T_p(n)]\left(1 - \rho_0 \frac{q(n)}{n}\right) + \mathbb{E}[T_s(n)]} + \frac{\mathbb{E}[T_p(n)] \cdot C_{par}}{n\mathbb{E}[T_p(n)]\left(1 - \frac{\rho_0}{n}\right) + \mathbb{E}[T_s(n)]} \right. \\ &\left. + \frac{\mathbb{E}[T_s(n)]}{n\mathbb{E}[T_p(n)] + \mathbb{E}[T_s(n)](1-\rho_0)} \right) \end{aligned} \tag{4.37}$$

Note that

$$\begin{aligned} \mathbb{E}[T_p(n)] &= \frac{\mathbb{E}[T_p(1)]h(n)}{n} \\ \mathbb{E}[T_s(n)] &= \mathbb{E}[T_s(1)]v(n) \\ \eta &= \frac{\mathbb{E}[T_p(1)]}{\mathbb{E}[T_p(1)] + \mathbb{E}[T_s(1)]} \\ \epsilon(n) &= \frac{h(n)}{v(n)} = \alpha n^{\delta} \qquad \text{with } \alpha > 0 \text{ and } \delta \leq 1 \end{aligned}$$

Substituting into Eq. (4.37) and divide both numerator and denominator of each term in the denominator of Eq. (4.37) by $(\mathbb{E}[T_p(1)] + \mathbb{E}[T_s(1)])v(n)$, the response time speedup can be rewritten as follows,

$$\begin{aligned} S_d(n) = &\left(1 + \frac{\rho_0}{1-\rho_0} \cdot \frac{1+C_s^2}{2}\right) \Bigg/ \\ &\left(\frac{\eta\alpha n^{\delta} \cdot \frac{q(n)}{n}}{\eta\alpha n^{\delta}\left(1 - \rho_0 \frac{q(n)}{n}\right) + (1-\eta)} + \frac{\eta\alpha n^{\delta} \cdot \frac{C_{par}}{n}}{\eta\alpha n^{\delta}\left(1 - \frac{\rho_0}{n}\right) + (1-\eta)} \right. \\ &\left. + \frac{(1-\eta)}{\eta\alpha n^{\delta} + (1-\eta)(1-\rho_0)} \right) \end{aligned} \tag{4.38}$$

With this general speedup, one can explore the scaling properties of a system on its entire design space. Here we present insights of several limiting cases from this general speedup.

Case 1: $\rho_0 = 0$

This corresponds to the case when there is no queuing at all the nodes in the system. One can easily see that Eq. (4.38) degenerates to

$$\begin{aligned} S_d(n) &= \frac{\eta\alpha n^\delta + (1-\eta)}{\eta\alpha n^\delta \cdot \dfrac{q(n)}{n} + \eta\alpha n^\delta \cdot \dfrac{C_{par}}{n} + (1-\eta)} \\ &= \frac{\eta\alpha n^\delta + (1-\eta)}{\eta\alpha n^\delta \left(\dfrac{q(n)}{n} + \dfrac{\mathcal{H}_n}{n}\right) + (1-\eta)} \end{aligned} \tag{4.39}$$

whose scaling properties has been studied in [39].

We now study the effect of the in-proportion or internal scaling without the scale-out-induced scaling, i.e., $q(n) = 0$. The scaling properties of these cases are shown as three solid curves in Fig. 4.4 as explained below.

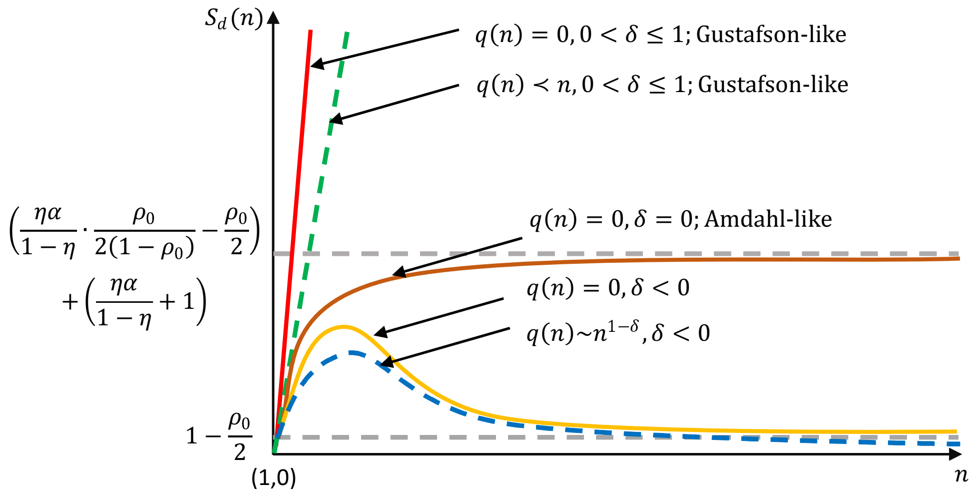


Figure 4.4: The scaling properties.

Case 2: $q(n) = 0$; $0 < \delta \leq 1$

The response time speedup can be rewritten as,

$$S_d(n) = \left(1 + \frac{\rho_0}{1-\rho_0} \cdot \frac{1+C_s^2}{2}\right) \Bigg/ \left(\frac{\eta \alpha n^\delta \cdot \dfrac{C_{par}}{n}}{\eta \alpha n^\delta \left(1 - \dfrac{\rho_0}{n}\right) + (1-\eta)} + \frac{(1-\eta)}{\eta \alpha n^\delta + (1-\eta)(1-\rho_0)} \right). \tag{4.40}$$

Due to the barrier synchronization, i.e., the scaling factor C_{par}, this speedup scales with n slower than the one stated out in the Gustafson's law but still sublinearly fast and unbounded since C_{par} grows "slower" than n and

$$\lim_{n\to\infty} S_d(n) = \left(1 + \frac{\rho_0}{1-\rho_0} \cdot \frac{1}{2}\right) \cdot \frac{n}{C_{par}} = \infty \,.$$

One can see that when n is large, the effect of ρ_0 on the two terms in the denominator of Eq. (4.40) is very small, i.e., not much different from the case without queuing, while factor $\frac{\rho_0}{1-\rho_0}$ in the nominator has a larger impact when ρ_0 becomes large, e.g., when $\rho_0 = 0.99$, $\frac{\rho_0}{1-\rho_0} = 99$. This mean that the speedup with queuing effect grows faster than that without queuing effect when ρ_0, equivalently the throughput, becomes large.

Result 1: *Without scale-out-induced scaling, the speedup is unbounded for in-proportion scaling with $0 < \delta \leq 1$ and grows faster than that for the case without queuing effect when the throughput becomes large.*

Case 3: $q(n) = 0$; $\delta = 0$

This represents the case where the external and internal scaling, i.e., $h(n)$ and $v(n)$, grow at the same order of "rate". The response time speedup can be rewritten as,

$$S_d(n) = \left(1 + \frac{\rho_0}{1-\rho_0} \cdot \frac{1+C_s^2}{2}\right) \Bigg/ \left(\frac{\eta\alpha \cdot \frac{C_{par}}{n}}{\eta\alpha\left(1 - \frac{\rho_0}{n}\right) + (1-\eta)} + \frac{(1-\eta)}{\eta\alpha + (1-\eta)(1-\rho_0)} \right). \tag{4.41}$$

We have,

$$\begin{aligned}\lim_{n\to\infty} S_d(n) &= \frac{\left(1 + \frac{\rho_0}{1-\rho_0} \cdot \frac{1}{2}\right)}{\frac{(1-\eta)}{\eta\alpha + (1-\eta)(1-\rho_0)}} \\ &= \left(1 + \frac{\rho_0}{2(1-\rho_0)}\right)\frac{\eta\alpha}{1-\eta} + \left(1 - \frac{\rho_0}{2}\right) \\ &= \left(\frac{\eta\alpha}{1-\eta} + 1\right) + \left(\frac{\eta\alpha}{1-\eta} \cdot \frac{\rho_0}{2(1-\rho_0)} - \frac{\rho_0}{2}\right).\end{aligned} \tag{4.42}$$

The speedup grows with increasing n but is eventually upper bounded by the term on the right-hand side of Eq. (4.42) as the case of Amdahl's law [7] but now depends not only on the serial portion but also the system throughput. The first term in the upper bound in Eq. (4.42) corresponds to the case without queuing, which degenerates to the Amdahl's law when $\alpha = 1$. With queuing, this upper bound may increase or decrease depending on ρ_0, equivalently the throughput. When $\rho_0 > \frac{1-\eta(1+\alpha)}{1-\eta}$, the bound increases and vice versa. For example, when $\eta = 0.9,\ \alpha = 0.1, and\, \rho_0 = 0.99$, the second term in Eq. (4.42) approximates 44.

Result 2: *Without scale-out-induced scaling, the speedup is upper-unbounded for in-proportion* *scaling at $\delta = 0$ and the upper bound is greater than that for the case without queuing effect when the throughput becomes large.*

Case 4: $q(n) = 0$; $\delta < 0$

We have,

$$\lim_{n\to\infty} S_d(n) = \frac{\left(1 + \dfrac{\rho_0}{1-\rho_0} \cdot \dfrac{1}{2}\right)}{\dfrac{1}{1-\rho_0}}$$
$$= 1 - \frac{\rho_0}{2} < 1 \qquad (4.43)$$

For this case, the speedup increases with n but eventually will drop back to a value less than one, depending on the system throughput. This differs from the case without queuing effect where this bound is one.

Result 3: *Without scale-out-induced scaling and for in-proportion scaling with $\delta < 0$ there exists a scale-out degree threshold, beyond which the speedup decreases and approaches a value less than one, which is the lower bound for the case without queuing effect.*

We now consider the effect of the scale-out-induced scaling on the speedup, i.e., when $q(n) > 0$. For the stability of the queuing model corresponding to the scale-out-induced stage, the utilization on this stage must be less than one, i.e.,

$$\begin{aligned}\frac{\rho_0\mathbb{E}[T_p(n)]q(n)}{n\mathbb{E}[T_p(n)]+\mathbb{E}[T_s(n)]} &< 1\\ \rho_0\eta\alpha n^\delta\cdot\frac{q(n)}{n} &< \eta\alpha n^\delta+(1-\eta)\\ q(n) &< \frac{n}{\rho_0}+\frac{(1-\eta)}{\rho_0\eta\alpha}\cdot n^{1-\delta}\end{aligned}\tag{4.44}$$

This suggests that $q(n)$ should not scale faster than n when $\delta \geq 0$ or faster than $n^{1-\delta}$ when $\delta < 0$. Not complying with this condition makes the scale-out-induced stage become a bottleneck, meaning that the overhead due to scaling out dominates the scaling gain and pulls the speedup down to zero quickly.

Case 5: $q(n) > 0$; $\delta > 0$

If $q(n)$ scales at the order of "rate" smaller than n, symbolized by $q(n) \prec n$, we have,

$$\lim_{n\to\infty} S_d(n) = \left(1+\frac{\rho_0}{1-\rho_0}\cdot\frac{1}{2}\right)\cdot\frac{n}{q(n)+C_{par}} = \infty \tag{4.45}$$

With the effect of the scale-out-induced stage, the speedup is pulled down a bit but still grows unboundedly with increasing n, as shown in the linear dashed line in Fig. 4.4.

If $q(n)$ scales at the same order of "rate" as n, symbolized by $q(n) \sim n$, the scale-out-induced scaling has an effect similar to a sequential portion, which places an upper bound on the speedup. This case is similar to case 3 above and is not shown in Fig. 4.4.

Case 6: $q(n) > 0$; $\delta < 0$

If $q(n)$ scales at the order of "rate" less than $n^{1-\delta}$, symbolized by $q(n) \prec n^{1-\delta}$, from Eq. (4.38), the effect of $q(n)$ is the same as that of C_{par}. As a result, the speedup is upper bounded by the serial portion as case 3 above.

If $q(n)$ scales at the same order of "rate" as $n^{1-\delta}$, symbolized by $q(n) \sim n^{1-\delta}$, the speedup grows with n but beyond a certain point, it decreases and approaches a lower bound when n becomes large. This bound is even less than $\left(1 - \frac{\rho_0}{2}\right)$ as in case 4. This speedup is illustrated as the dashed curve in Fig. 4.4.

Result 4: *With $\delta < 0$ and $q(n) \sim n^{1-\delta}$, there exists a scale-out degree threshold, beyond which the speedup decreases and approaches zero.*

The derivation of the throughput speedup in Eq. (4.28) is much more involved. We leave it for our future work. Figs. 4.5–4.7 show the throughput speedup numerically with several different experimental settings. Scenario 1's settings are similar to the case of Gustafson's law plus a small effect of the scale-out-induced stage. As with Gustafson's law, we obtain a linear speedup for the throughput in this case. Scenario 2 has the same setting as scenario 1 but with more effect on the scale-out-induced scaling. One can see that the throughput speedup is upper bounded when n becomes large. In scenario 3, we assume the scale-out-induced scaling grows with a rate greater than n. As a result, there exist certain thresholds beyond which the speedup starts dropping.

4.4 Conclusions

In this chapter, we put forward a framework for scaling performance analysis of datacenter applications. It took into account the effect of overhead due to scaling out, referred to as the

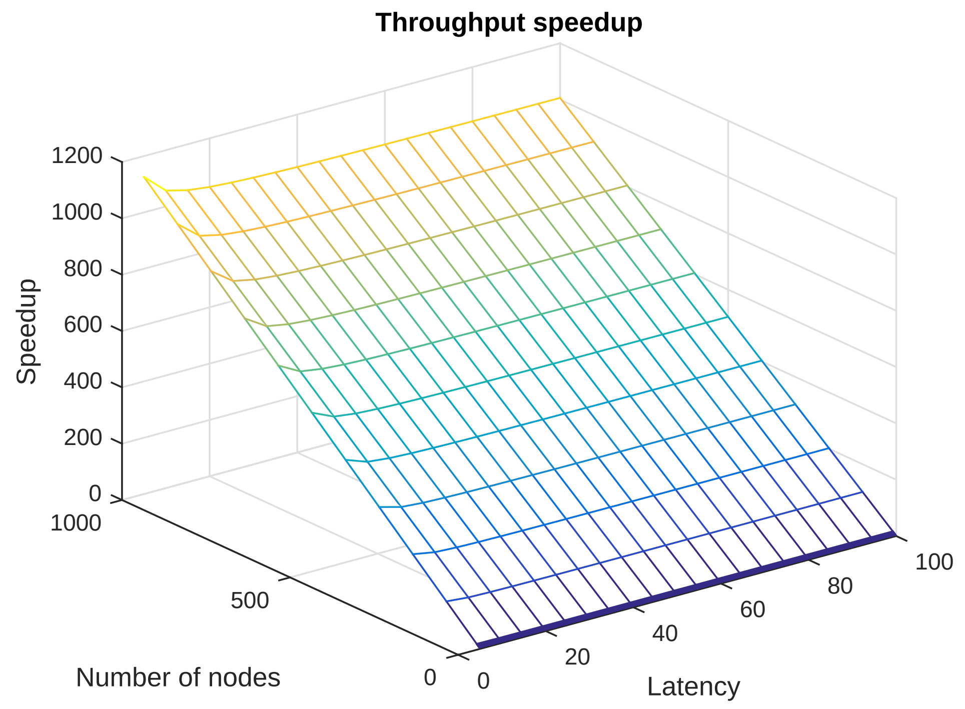


Figure 4.5: Scenario 1: $h(n) = n$, $v(n) = 1$, and $q(n) = 0.001n$.

scale-out-induced scaling and the scaling of the serial portion of the workload, referred to as the *in-proportion scaling*. The framework generalized the work in [39] to deal with the case when there is queuing effect at each node in the system. We showed the effect of queuing on scaling factors as well as the bounds of the speedup through several limiting case studies. This is a work in progress. We presented here an analytical solution for the response time speedup for a special case when all the queues have exponential service times. In our future work, we plan to explore the framework with other types of service time distribution and test it numerically through simulation-based and measurement-based systems. We also plan to derive an analytical solution for the throughput speedup.

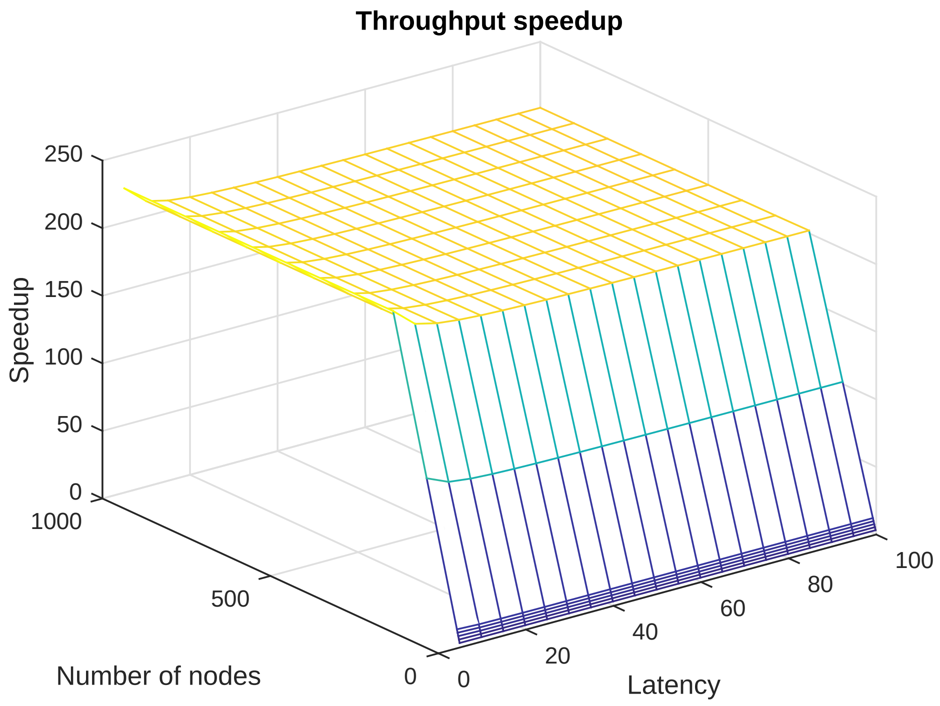


Figure 4.6: Scenario 2: $h(n) = n$, $v(n) = 1$, and $q(n) = 0.005n$.

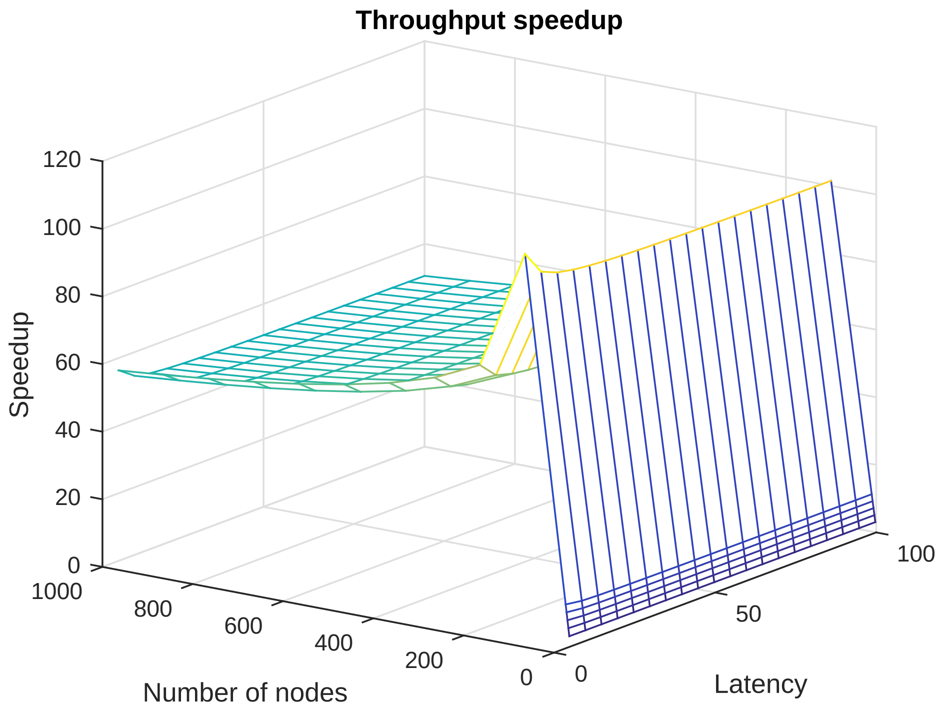


Figure 4.7: Scenario 3: $h(n) = n$, $v(n) = 1$, and $q(n) = 0.005n^{1.2}$.

Chapter 5

CONCLUSIONS AND FUTURE WORK

This dissertation developed approximation solutions for prediction of the tail and mean latency of the Fork-Join queuing network, a critical structure in the workflow of scale-out applications. The Fork-Join models are known to be hard to analyze due to the involvement of task partitioning and merging with barrier synchronization. The proposed models specifically targeted at high load region, where resource provisioning is desirable. The accuracy of the tail and mean latency predictions were shown through extensive experiments with different settings, which yields errors consistently within 10% at the load of 90%. Hence the proposed approximations provide viable tools for resource provisioning of scale-out applications with multiple constraints, i.e., multiple SLOs.

The dissertation also introduced a framework for scaling analysis of scale-out applications. With the proposed models, closed-form solutions were found, upon which rich scaling properties for scale-out applications can be explored. A case study with exponential task service times is also presented.

Our future work will focus on further testing and perfecting the proposed solutions including,

– Testing the tail and mean latency prediction models in a testbed for various scale-out applications.

– Developing SLO-guaranteed job scheduling and resource provisioning algorithms based on the proposed solutions.

– Deriving an analytical solution for the throughput speedup for scaling analysis as well as carrying out simulation-based and measurement-based testing for the framework.

Appendix A

MORE RESULTS FOR TAIL LATENCY PREDICTION IN CHAPTER 3

Table A.1: Exponential distribution: Errors in the 99th percentiles

	Number of nodes									
Load	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
50%	2.156	2.466	2.432	2.951	3.087	3.275	3.213	3.184	3.390	3.542
75%	3.240	3.973	4.900	4.849	5.174	5.283	4.466	5.095	5.883	5.365
80%	3.647	5.226	4.182	5.149	5.893	4.485	5.637	5.472	6.053	5.430
90%	6.214	6.927	4.247	5.948	6.766	6.569	5.860	5.828	6.746	6.799

Table A.2: Exponential distribution: Errors in the 99.9th percentiles

	Number of nodes									
Load	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
50%	0.781	0.303	0.462	1.072	1.176	1.283	1.098	0.747	1.141	1.341
75%	1.108	1.319	1.960	1.298	1.316	1.442	0.644	2.654	2.571	1.593
80%	1.975	2.912	1.103	2.513	3.801	0.509	2.598	1.608	3.416	2.171
90%	3.412	3.168	1.411	2.624	4.111	3.583	2.110	2.924	3.029	3.802

Table A.3: Gamma distribution: Errors in the 99th percentiles

	Number of nodes									
Load	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
50%	2.373	2.873	3.231	3.703	4.042	3.975	3.911	4.052	4.443	4.395
75%	4.977	4.893	6.797	6.309	7.295	7.539	7.084	7.476	6.899	7.925
80%	6.182	6.333	6.756	7.435	8.363	6.982	8.340	7.972	8.500	9.338
90%	7.318	7.352	8.695	9.565	9.090	10.301	11.704	10.672	9.421	10.722

Table A.4: Gamma distribution: Errors in the 99.9th percentiles

	Number of nodes									
Load	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
50%	-0.062	-0.444	0.325	0.643	0.886	0.749	0.353	0.090	1.220	1.025
75%	2.720	0.565	2.247	1.973	2.204	3.313	0.384	1.926	1.124	3.258
80%	2.494	0.546	2.358	3.450	3.432	1.831	4.476	2.034	3.971	4.686
90%	2.833	-2.801	3.869	1.715	3.496	3.852	6.938	5.529	1.393	2.649

Table A.5: Weibull distribution: Errors in the 99th percentiles

	Number of nodes									
Load	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
50%	-2.920	-3.181	-3.401	-3.640	-3.811	-3.854	-3.809	-3.906	-3.915	-3.831
75%	2.458	2.759	3.161	2.874	2.959	3.397	3.203	2.995	3.356	3.188
80%	3.138	3.968	3.315	3.581	3.456	3.000	3.334	2.532	4.024	3.597
90%	2.411	4.155	2.806	3.528	3.906	1.731	2.946	4.062	4.501	3.168

Table A.6: Weibull distribution: Errors in the 99.9th percentiles

	Number of nodes									
Load	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
50%	-4.202	-4.372	-5.131	-4.876	-5.011	-5.252	-5.237	-5.459	-4.750	-5.291
75%	2.199	2.835	2.162	0.665	3.003	3.351	1.938	2.396	2.382	1.560
80%	2.337	4.374	2.050	2.124	2.899	2.103	1.483	1.711	3.386	2.020
90%	2.790	5.735	3.927	0.734	2.412	-1.028	3.799	4.796	5.333	0.591

Table A.7: Truncated Pareto distribution: Errors in the 99th percentiles

	Number of nodes									
Load	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
50%	-34.300	-32.360	-31.420	-31.359	-31.308	-31.654	-31.199	-31.718	-31.687	-32.008
75%	-14.476	-14.842	-14.768	-14.732	-15.205	-14.936	-15.057	-15.236	-15.083	-14.916
80%	-10.078	-10.377	-11.229	-10.677	-11.008	-10.558	-10.689	-10.842	-10.131	-10.754
90%	-1.157	-0.974	-1.263	-1.025	-1.888	-0.047	-1.010	-0.553	-0.765	-1.769

Table A.8: Truncated Pareto distribution: Errors in the 99.9th percentiles

	Number of nodes									
Load	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
50%	-32.559	-33.060	-33.806	-33.995	-34.230	-34.772	-33.508	-34.759	-34.314	-34.500
75%	-14.714	-15.475	-15.453	-15.278	-16.604	-14.997	-14.242	-15.442	-14.750	-15.324
80%	-9.770	-9.785	-11.562	-11.243	-11.025	-10.159	-10.485	-10.696	-10.769	-11.668
90%	-1.586	-1.912	-2.847	-0.941	-1.808	-0.869	0.915	-0.961	-2.063	-3.171

Table A.9: Empirical distribution: Errors in the 99th percentiles

	Number of nodes									
Load	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
50%	-42.100	-38.937	-37.564	-37.087	-36.759	-36.854	-36.763	-37.375	-37.034	-37.391
75%	-18.368	-20.436	-20.826	-20.112	-20.914	-21.482	-20.422	-21.120	-21.825	-21.024
80%	-14.037	-14.294	-15.501	-15.381	-15.179	-15.232	-15.444	-15.587	-15.121	-15.775
90%	1.614	2.874	1.104	2.121	2.288	1.733	3.165	2.739	0.107	2.425

Table A.10: Empirical distribution: Errors in the 99.9th percentiles

	Number of nodes									
Load	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
50%	-38.811	-39.394	-39.774	-40.732	-41.038	-40.679	-40.912	-41.990	-41.485	-41.599
75%	-21.456	-22.925	-22.007	-21.142	-20.406	-22.810	-20.056	-21.733	-21.966	-19.812
80%	-15.888	-13.744	-16.148	-13.995	-14.253	-15.315	-14.393	-15.417	-14.999	-14.788
90%	1.053	4.234	1.766	4.207	1.673	1.615	6.465	0.649	0.598	1.925

BIBLIOGRAPHY

[1] Apache Hadoop. `https://hadoop.apache.org`.

[2] Apache Spark. `https://spark.apache.org`.

[3] Discrete Triangular Distribution. `https://www.me.utexas.edu/~jensen/ORMM/computation/unit/rvadd/discrete_dist/triangular.html`.

[4] The M/M/1 Fork-Join Queue with Variable Sub-Tasks. `http://www.cs.unh.edu/~varki/publication/2002-nov-open.pdf`.

[5] I. B. Aban, M. M. Meerschaert, and A. K. Panorska. Parameter Estimation for the Truncated Pareto Distribution. *Journal of the American Statistical Association*, 101(473):270–277, Mar. 2006.

[6] 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*, 25(6):1437–1446, 2014.

[7] G. M. Amdahl. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In *Proceedings of American Federation of Infomation Processing Societies Conference*, pages 483–485, 1967.

[8] S. Balsamo, L. Donatiello, and N. Van Dijk. Bound Performance Models of Heterogeneous Parallel Processing Systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(10):1041–1056, 1998.

[9] S. Balsamo and I. Mura. Approximate Response Time Distribution in Fork and Join Systems. In *Proceedings of the 1995 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '95/PERFORMANCE '95, pages 305–306, 1995.

[10] L. A. Barroso, J. Clidaras, and U. Hölzle. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition. *Synthesis Lectures on Computer Architecture*, 8(3):1–154, July 2013.

[11] L. A. Barroso, J. Dean, and U. Hölzle. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro*, 23(2):22–28, Mar. 2003.

[12] L. A. Barroso and U. Hölzle. The Case for Energy-proportional Computing. *Computer*, 40(12):33–37, 2007.

[13] 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.

[14] J. Brutlag. Speed Matters for Google Web Search. `https://services.google.com/fh/files/blogs/google_delayexp.pdf`, 2009.

[15] G. Brys, M. Hubert, and A. Struyf. Robust Measures of Tail Weight. *Computational Statistics & Data Analysis*, 50(3):733–759, feb 2006.

[16] H. Che and M. Nguyen. Amdahl's Law for Multithreaded Multicore Processors. *Journal of Parallel and Distributed Computing*, 74(10):3056–3069, Oct. 2014.

[17] R. Chen. A Hybrid Solution of Fork/Join Synchronization in Parallel Queues. *IEEE Transactions on Parallel and Distributed Systems*, 12(8):829–845, 2001.

[18] R. J. Chen. An Upper Bound Solution for Homogeneous Fork/Join Queuing Systems. *IEEE Transactions on Parallel and Distributed Systems*, 22(5):874–878, May 2011.

[19] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing Data Transfers in Computer Clusters with Orchestra. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 98–109, 2011.

[20] J. Dean and L. A. Barroso. The Tail at Scale. *Communications of the ACM*, 56(2):74–80, Feb. 2013.

[21] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, OSDI '04, pages 137–150, 2004.

[22] C. Delimitrou and C. Kozyrakis. Quasar: Resource-efficient and QoS-aware Cluster Management. *ACM SIGARCH Computer Architecture News*, 42(1):127–144, Apr. 2014.

[23] S. Eyerman and L. Eeckhout. Modeling Critical Sections in Amdahl's Law and Its Implications for Multicore Design. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 362–370, 2010.

[24] A. D. Ferguson, P. Bodik, E. Boutin, and R. Fonseca. Jockey : Guaranteed Job Latency in Data Parallel Clusters. In *Proceedings of the 7th ACM European Conference on Computer Systems*, pages 99–112, 2012.

[25] L. Flatto and S. Hahn. Two Parallel Queues Created by Arrivals with Two Demands I. *SIAM Journal on Applied Mathematics*, 44(5):1041–1053, Oct. 1984.

[26] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytiä, and A. Scheller-wolf. Reducing Latency via Redundant Requests: Exact Analysis. In *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '15, pages 347–360, 2015.

[27] R. D. Gupta and D. Kundu. Generalized Exponential Distributions. *Australian & New Zealand Journal of Statistics*, 41(2):173–188, 1999.

[28] J. L. Gustafson. Reevaluating Amdahl's Law. *Communications of the ACM*, 31(5):532–533, 1988.

[29] M. E. Haque, Y. hun Eom, Y. He, S. Elnikety, R. Bianchini, and K. S. McKinley. Few-to-Many: Incremental Parallelism for Reducing Tail Latency in Interactive Services. In *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, pages 161–175, 2015.

[30] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A Platform for Fine-grained Resource Sharing in the Data Center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI '11, pages 295–308, 2011.

[31] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 59–72, 2007.

[32] V. Jalaparti, P. Bodik, S. Kandula, I. Menache, M. Rybalkin, and C. Yan. Speeding Up Distributed Request-Response Workflows. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 219–230, 2013.

[33] M. Jeon, S. Kim, S.-w. Hwang, Y. He, S. Elnikety, A. L. Cox, and S. Rixner. Predictive Parallelization: Taming Tail Latencies in Web Search. In *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '14, pages 253–262, 2014.

[34] J. F. C. Kingman and M. F. Atiyah. The Single Server Queue in Heavy Traffic. *Proceedings of the Cambridge Philosophical Society*, 57:902–904, 1961.

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

[36] S.-S. Ko and R. F. Serfozo. Response Times in M/M/s Fork-Join Networks. *Advances in Applied Probability*, 36(3):854–871, 2004.

[37] J. Köllerström. Heavy Traffic Theory for Queues with Several Servers. I. *Journal of Applied Probability*, 11(3):544–552, 1974.

[38] A. Lebrecht and W. J. Knottenbelt. Response Time Approximations in Fork-Join Queues. In *23rd Annual UK Performance Engineering Workshop (UKPEW)*, June 2007.

[39] Z. Li, F. Duan, M. Nguyen, H. Che, Y. Lei, and H. Jiang. IPSO: A New Scaling Model for Data-Intensive Applications, (*In submission*).

[40] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis. Towards Energy Proportionality for Large-scale Latency-critical Workloads. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, pages 301–312, 2014.

[41] M. Manivannan, B. Juurlink, and P. Stenstrom. Implications of Merging Phases on Scalability of Multi-core Architectures. In *International Conference on Parallel Processing (ICPP)*, pages 622–631, 2011.

[42] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 248–259, Dec. 2011.

[43] D. Meisner, W. Junjie, and T. F. Wenisch. BigHouse: A Simulation Infrastructure for Data Center Systems. In *Proceedings of the 2012 IEEE International Symposium on Performance Analysis of Systems & Software*, ISPASS '12, pages 35–45, 2012.

[44] D. Meisner, C. M. Sadler, L. B. Andre, 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*, ISCA '11, pages 319–330, 2011.

[45] R. Nelson and A. Tantawi. Approximate Analysis of Fork/Join Synchronization in Parallel Queues. *IEEE Transactions on Computers*, 37(6):739–743, June 1988.

[46] 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.

[47] 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 *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, NSDI '13, pages 385–398, Apr. 2013.

[48] Z. Qiu and J. F. Perez. Evaluating the Effectiveness of Replication for Tail-Tolerance. In *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 443–452, May 2015.

[49] H. Qu, O. Mashayekhi, D. Terei, and P. Levis. Canary: A Scheduling Architecture for High Performance Cloud Computing. In *arXiv:1602.01412v1 [cs.DC]*, 2016.

[50] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and Dynamicity of Clouds at Scale. In *Proceedings of the 3rd ACM Symposium on Cloud Computing*, SoCC '12, pages 1–13, Oct. 2012.

[51] A. Rizk, F. Poloczek, and F. Ciucu. Computable Bounds in Fork-Join Queueing Systems. In *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '15, pages 335–346, 2015.

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

[53] X. H. Sun and Y. Chen. Reevaluating Amdahl's Law in the Multicore Era. *Journal of Parallel and Distributed Computing*, 70(2):183–188, 2010.

[54] L. Suresh, M. Canini, S. Schmid, and A. Feldmann. C3: Cutting Tail Latency in Cloud Data Stores via Adaptive Replica Selection. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, pages 513–527, May 2015.

[55] A. Thomasian. Analysis of Fork/Join and Related Queueing Systems. *ACM Computing Surveys*, 47(2):1–71, Aug. 2014.

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

[57] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. 2nd ed. Wiley, 2002.

[58] E. Varki. Response Time Analysis of Parallel Computer and Storage Systems. *IEEE Transactions on Parallel and Distributed Systems*, 12(11):1146–1161, 2001.

[59] S. Varma and A. M. Makowski. Interpolation Approximations for Symmetric Fork-Join Queues. *Performance Evaluation*, 20(1-3):245–265, May 1994.

[60] V. K. Vavilapalli, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, E. Baldeschwieler, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, and H. Shah. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, SoCC '13, pages 1–16, 2013.

[61] A. Vulimiri, J. Sherry, U. C. Berkeley, P. B. Godfrey, S. Ratnasamy, and S. Shenker. Low Latency via Redundancy. In *The 9th International Conference on emerging Networking EXperiments and Technologies*, CoNEXT '13, pages 283–294, 2013.

[62] 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*, SoCC '12, pages 14:1–14:14, 2012.

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

[64] T. Zhang, S. Kang, and L. Lipsky. On The Performance of Parallel Computers: Order Statistics and Amdahl's Law. *International Journal of Computers and Their Applications*, 3(2), 1996.

[65] 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*, SoCC '14, pages 1–14, 2014.