CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Performance evaluation and modeling of remote execution and caching cluster

Master's Thesis in Computer Systems and Networks

MARÍA FERNANDA AGUILAR ROMERO

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021

Master's thesis 2021

Performance evaluation and modeling of remote execution and caching cluster

MARÍA FERNANDA AGUILAR ROMERO

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2021

Performance evaluation and modeling of remote execution and caching cluster

MARÍA FERNANDA AGUILAR ROMERO

Supervisor: Marina Papatriantafilou, CSE
Advisor: Hans Hazelius, Veoneer
Examiner: Romaric Duvignau, CSE

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2021

Performance evaluation and modeling of remote execution and caching cluster
MARÍA FERNANDA AGUILAR ROMERO
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The scope and application of distributed systems are an increasing trend by the need for high-performance infrastructure. Many multi-branch organizations adopt the resource-sharing approach to maximize the system's utilization, especially in the software developing industry. Computational techniques such as caching and clustering are suitable to boost the performance of computer systems at any architectural level. This project presents a performance evaluation, modeling, and analysis of a system inspired by the case study of the industrial partner, Veoneer. The case study is used for software development and implements Bazel, a suitable tool for multi-branch cooperation through remote caching and remote execution. Here, the remote cache allows storage and sharing the outputs from all the compilations among the branches, thus preventing the re-execution of tasks and reducing the response time for developers. On the other hand, remote execution benefits from the computational resources of different servers across the branches. However, the benefits are at the cost of complexity, and systems start to degrade in performance, such as long response times revealed from the system's monitoring. Therefore, performance evaluation is essential for planning and improvement.
This work uses operational analysis, queueing networks, and the universal scalability law to evaluate, model, and predict the system's performance, respectively. The aim is to establish algebraic relations between the system's factors and metrics to understand the impact of workload and system capacity on performance to predict its behavior. The results expose that the cache is over-utilized and benefits from horizontal scalability. On the other hand, the study reveals that a cluster benefits from vertical scalability but still has its scalability bounds to prevent that multi-threading effects (i.e. coherence, concurrency, and contention) affect performance on parallel executions.

Keywords: Performance evaluation, performance modeling, remote cache, remote execution, queueing modeling, scalability.

Acknowledgements

First and foremost, I have to thank the Swedish Institute. Without its scholarship program for global professional leaders, this step in my professional career would have never been accomplished. I would also like to thank to my academic supervisor, Marina Papatriantafilou, and my advisor from Veoneer, Hans Hazelius, for their assitance, patience, and dedicated involvement in my learning path during the project.
I would also like to express my gratitude to my former professors, Patricia and Ignacio, and my former colleages, José and Gabriel, who trusted on my capacity and encouraged me to keep learning and growing.

Finally and most importantly, none of this could have happened without my family, who, despite of the distance and time difference, made my journey easier by providing me with unfailing support. Thank you.

María Fernanda Aguilar Romero, Gothenburg, November 2021

Contents

List of Figures

List of Tables

1

Introduction

The increasing trend of big data and computing-intensive applications need more complex, reliable, and scalable infrastructure. Behind the scenes, software development is a significant pillar for such applications, acting as a cycle, where application performance is a vital and meaningful metric of competitiveness in the software industry. Hence, clustering is an in-growing trend towards improving performance by grouping and sharing resources within the members of a cluster. Likewise nodes, that work together for either local or remote execution of tasks, cache instances are also able to group up and act as one network resource. Despite the enormous benefit from such load-balancing techniques, the largest the network, the hardest the performance evaluation; further, representing the system in a model implies understanding the impact that each element delivers. Thus, there is a need for an individual assessment of the components for a more accurate result [1].

Undoubtedly, performance evaluation, modeling, and analysis are crucial stages towards optimization and need to be performed from time to time, especially for capacity planning, bottlenecks avoidance, and upgrading. Performance of distributed systems depends on multiple factors such as the hardware characteristics (i.e., CPU, RAM, disk, network transmission media, capacity), quality of software management tools (e.g., scheduler, load-balancer), and workload (i.e., number of jobs, job arrival rate, job type). In most cases, performance deterioration is linear to workload since weak units start originating bottlenecks and impacting the overall performance of the system [2]. However, a sudden heavy-workload can also cause unexpected performance deterioration. All in all, performance turns around the load the system is required to process and the metrics selected to qualify the system's service.

Existing approaches for performance evaluation and modeling usually differ on the assessed target, the system configuration, and the system capacity features. For instance, distributed caching modeling [3], [4], network/traffic evaluation and modeling [5], [6], [7], distributed systems modeling [8], [9]. Notwithstanding, the available benchmarks are not useful and suitable for every system; they need adjustment to reflect their properties and allow minimum degrees of freedom for control tuning [10]. Evaluating performance is as meaningful as the measurement model's complexity. Therefore, there is a need for being aware of inputs and outputs while evaluating a system and tuning parameters to identify its behavior based on both load and metrics.

1.1 Context

Performance awareness starts from selecting the appropriate tools and techniques for project development. For instance, remote execution and caching are techniques to improve the performance of application deployment. Remote execution intends to offload local servers, whereas remote cache aims to reduce the number of compilations by reusing the artifacts from previous executions. Bazel and Buildbarn are open-source tools aiming to increase software development productivity by running build tasks using remote execution and caching [11].

As an illustration of the features previously described, Figure 1.1 shows how the remote execution and caching features of Buildbarn support application deployment. The first time, the system distributes the compilation task on the remote executors, saves the result in the cache, and returns the result to the client. From next time on, if the code to compile remains the same, Buildbarn retrieves the result from the cache; otherwise, remote workers compile what is new (i.e., last changes in the code) and retrieves the results from previous compilations to produce the ultimate result. Hence, the client expects to experience faster compilations as a result of using the cache. The frontend (scheduler) is for illustration purposes only.

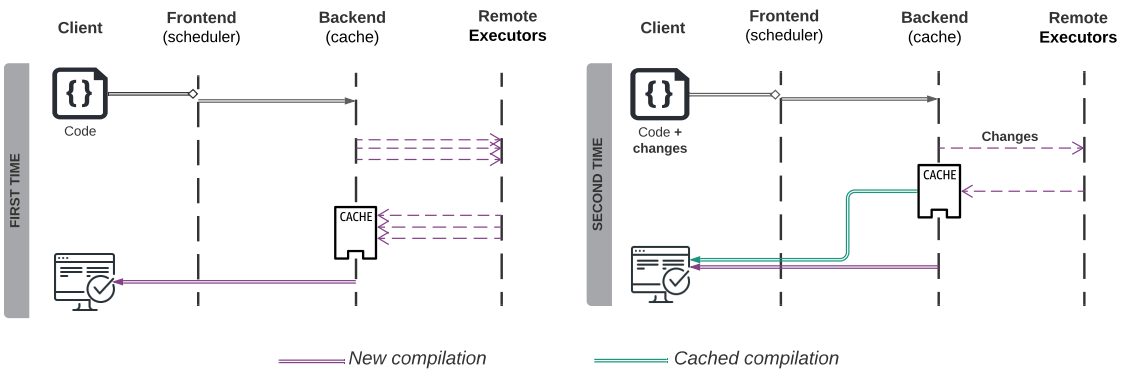
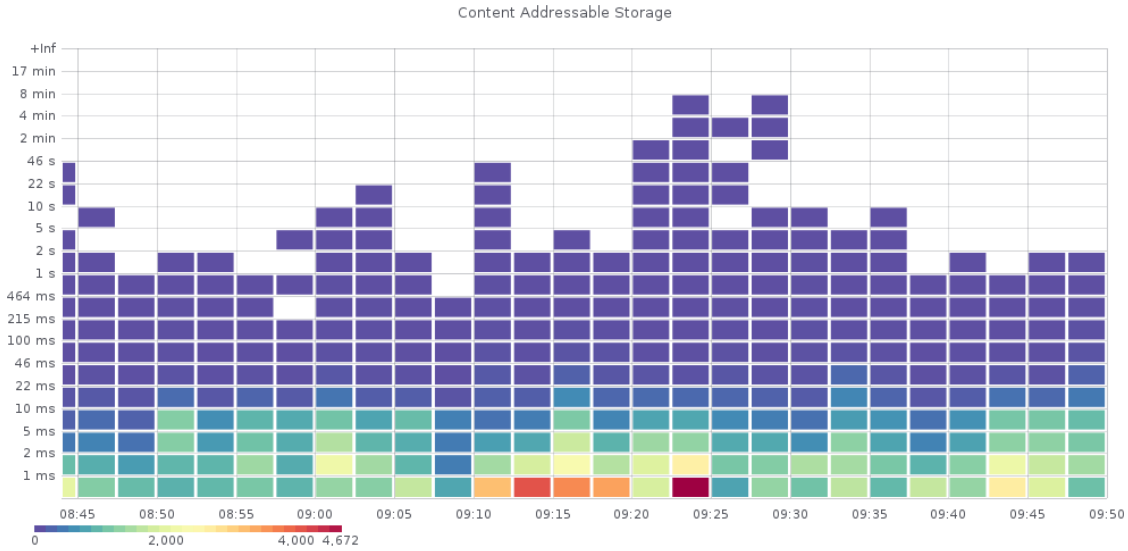


Figure 1.1: Overview of expected operation of a system that implements remote execution and caching services.

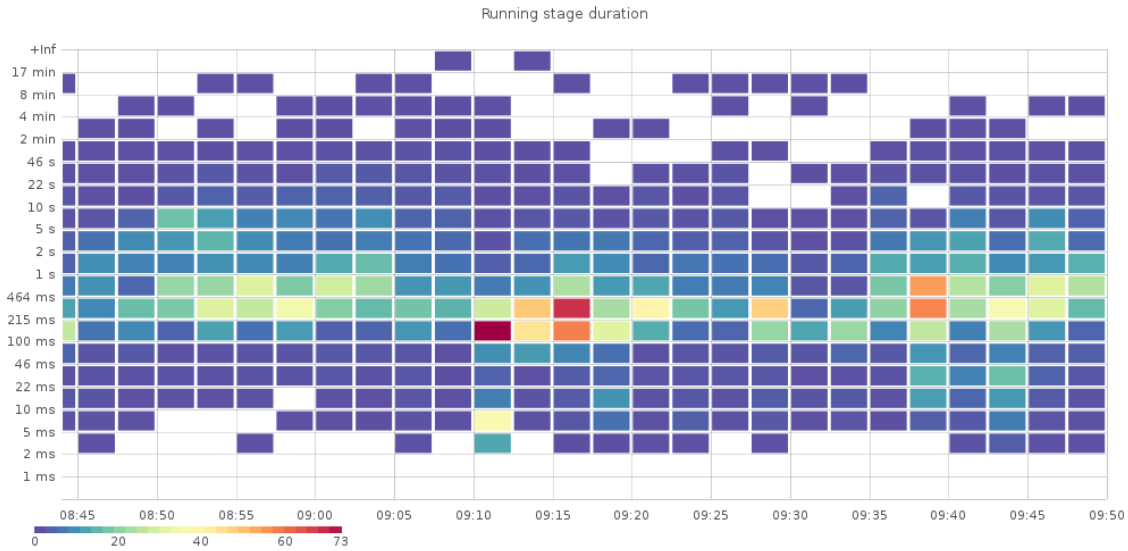
This thesis presents the system of the main case study, which corresponds to the *Test Infrastructure and Management* production system by Veoneer. The system relies on Bazel to provide of remote caching and remote execution features to the multi-branch organization. Veoneer also provided the required infrastructure for testing, as Chapter 3 describes.

Although there is a detailed description of the system in Chapter 2, roughly speaking, it works as follows: the requests from clients arrive at the Action Cache (AC) for it to determine whether the required content is in the cache or not; if so, the system retrieves the result from the cache, otherwise, the request is sent to the workers for execution. In the end, the response time corresponds to both services' response time. As for illustration, Figure 1.2 shows the service-time density distribution to

retrieve content from the cache (a) and executing an action on the workers (b) under a regular one-week workload. The figure plots the timeline on the x axis and the execution duration, which ranges from milliseconds (ms) to minutes (min) on the y axis. The job density distribution is colored from low (*blue*) to high (*red*) according to the number of jobs that arrived during the time interval of the x axis. For instance, a high number of jobs with a GET duration of 1ms or less occurred around 9:15 a.m. in the cache, whereas a high number of jobs with a running duration between 215ms and 464ms occurred in the cluster the same time.



(a) Cache-GET Requests



(b) Remote Execution Cluster

Figure 1.2: Density distribution of job duration in the cache and the remote execution cluster.

Figure 1.2 shows that the higher density of cached jobs is served between 0 and 2

milliseconds, whereas remote workers serve their majority of jobs between 46 and 215 milliseconds. Likewise, the number of jobs spending more than two minutes is higher in the remote execution than the cache. Although the cache system is significantly faster than the remote execution system, there is no infinite storage. Therefore, this project's performance analysis is also oriented to find a suitable configuration for the remote executors to improve their impact on the response time.

Like the execution time, the existing system provides accurate measures for problem identification; however, it is challenging to perform any hardware or software improvement in production environments. Therefore, there is a need for a performance model that conceives tunable inputs and outputs to assess the goals. Besides, there is a need for scenarios to validate the model.

Since there is not a *default* model to characterize distributed systems, it varies according to the systems configuration and the available resources to conduct the assessment. Frincu et al. summarize different models for infrastructure representation, based on simulations of various scenarios [9]. Kattepur et al. present a single-user test approach, which differs from the workload approach in the accuracy of bottlenecks identification caused by resource utilization, instead of traffic load [7]. Hence, the workload approach is the most suitable one for the evaluation this work aims to perform.

1.2 Goals and Challenges

This project aims to characterize the system, object of study, by describing it through an analytical model that eases the identification of weaknesses when deploying applications. To be more specific, the intention is to estimate a model to follow the bottlenecks' path and find the elements with the most impact on performance (i.e., network bandwidth, processor, memory, cache) according to different workloads, and considering the interaction between the components.

Indeed, modeling and simulation are complementary approaches to understanding systems behavior and to get significant insights about their performance. However, computational resources are not always available to emulate an entire infrastructure and its complexity. Moreover, there is a risk of tuning parameters within the production environment because of unexpected real-time operation effects. Therefore, this project experimented in a lab environment that replicates the production system corresponding to the case study but is shorter.

Another challenge is the number of elements that are part of the study. At first sight, every component within the infrastructure is an object of performance analysis due to its impact on the execution path. Notwithstanding, the goal of modeling is to focus on the sections under the Buildbarn management and configuration, it is, the interaction between *frontend*, *backend*, *cache*, and *workers*. Also, even when performance modeling leads to optimization, the latter is not part of the project since such improvements are highly dependent on the model results and the company's

decisions.

1.3 Outline

The thesis proceeds with a background description in Chapter 2, which reviews fundamental aspects of performance evaluation and modeling. Following this, Chapter 3 describes the methodology by discussing the assumptions, the theoretical model, and the experimental setup for data gathering. The results from evaluation and modeling are presented in Chapter 4. Finally, Chapter 5 concludes the work by summarizing and discussing the findings of the project, as well as presenting open questions fo further work.

2
Background

In essence, performance is the quantitative measurement of a service provided by a system. That is why it is crucial to carry out performance evaluation and modeling when designing and running a system. In this chapter, Section 2.1 reviews the concepts applied to the project. Then, there is a summary of performance evaluation in section 2.2, and system modeling in section 2.3. Finally, Section 2.4 shows relevant work, from the literature review, about performance modeling and evaluation in distributed systems and networks. The following compilation has been oriented to the system of the case study for a better understanding.

2.1 Concepts

Undoubtedly, a system's performance relies upon optimal configurations of both software and hardware. Although the latter is harder to improve quickly on production environments, policies and arrangements on key components might significantly upgrade a system's operation. Distribute systems platforms are widely applied on high-performance structures due to their reliability, controllability, and scalability [8]. This project refers the performance evaluation of a system that implements two of the most efficient techniques for high-performance distributed systems: clustering and caching, which are explained in the sections below.

2.1.1 Distributed Systems

The concept of *distributed system* varies in the literature in line with the context. However, and without loss of generality, a distributed system is a group of independent instances represented as one. In principle, distributed systems aim for collaborative work by summing up the individual effort from each instance to meet the demand [12]. Nonetheless, when grouping instances to work together, there is an essential trade-off between sharing and coherence among instances.

Illustrated with Figure 2.1 is the distributed systems' concept. Fundamentally, it requires, at least, a client, a service, and a communication network, intending to share resources. Misconfigurations in any of the components might lead to issues that impact the performance of the entire system. For instance, a high number of users, small system capacity, and low interconnection bandwidth. Thus, performance analysis of distributed systems allows identifying the weaknesses of the systems and so amend them. Likewise, solutions range from process improvement

to hardware upgrades (e.g., network bandwidth increase, system capacity extension, incoming requests limitation).

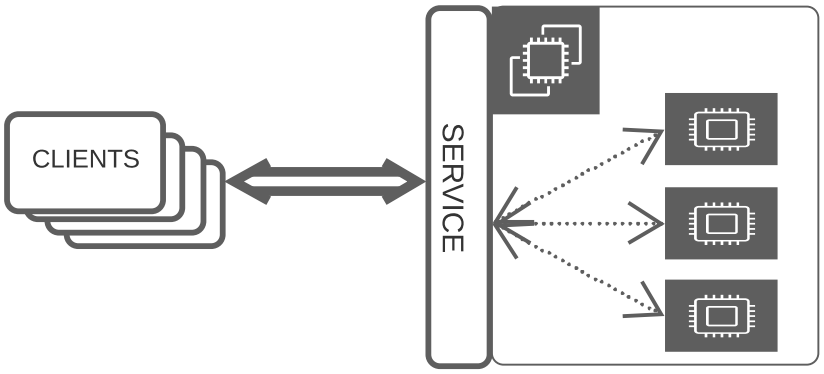


Figure 2.1: Overview of a client-server model of distributed system.

Clustering is the proper name for grouping resources. In computer systems, a cluster might be a group of computers, threads, or even memory pages for non-uniform address spaces [13]. Clustering gives way for high-performance techniques such as fair distribution and load balancing among the grouped instances, leading to consider clustering as exceptionally advantageous for parallelization. Hence, it becomes crucial to consider scalability when designing a distributed system, so the available resources increase as the demand grows.

Scalability is a concept used to measure the return of investment. There are different dimensions of scalability for distributed systems. For example, one can think about increasing the system's size (i.e., more instances, more computational resources), its geographical scope (i.e., expand the system to different locations by network), or its administration features (i.e., guarantee management from different dependencies). In most cases, scalability is required to overcome performance issues due to capacity limits [13]. Ideally, it is expected that doubling the number of nodes in a cluster, would increase twice the capacity of the system. However, linear scalability is not often delivered. The main reasons the distributed systems are vulnerable to scale are: *contention*, *coherence*, and *concurrency* [14].

Therefore, solutions should demonstrate that improvements are proportional to performance gain; otherwise, it can be said the system is not scalable. There are two main approaches to add capacity in computer systems for processing. On the one hand, horizontal scalability refers to increase the number of nodes in the system. On the other hand, vertical scalability increases the computational resources (e.g., RAM, number of processors, disk size) per node. What approach delivers better performance depends on the system configuration and workload type.

Moreover, there are different communication approaches for distributed systems depending on their architecture. A centralized approach, commonly known as the server-client model, designates an instance to receive and handle the requests. Instead, a decentralized system's organization, commonly known as peer-to-peer

model, splits the resources for any of the instances with the required resource handle the request. Therefore, two primary communications approaches are suitable to communicate users and applications on distributed systems. On the one hand, the message passing paradigm that relies on message operations where clients send the whole stack to the server for processing. On the other hand, the remote procedure call that relies on procedures inquiries with no further details about the underlying software implementation. The latter suits better for client-server architecture [12], which also gives way to remote execution schemes as the case study of this project and is explained below.

2.1.2 Caching

The caching concept varies according to the system's architectural level it belongs. Nevertheless, a broader description of cache summarizes it as the mechanism that allows reusing information through buffering [15]. As the records in a cache are replaced when needed, every successful finding is called *hit*; otherwise, if the lookup fails, it is called *miss*. The cache's primary asset is to prevent the system from double processing. It is, avoiding the execution of actions already executed under the same conditions with a result already produced and stored.

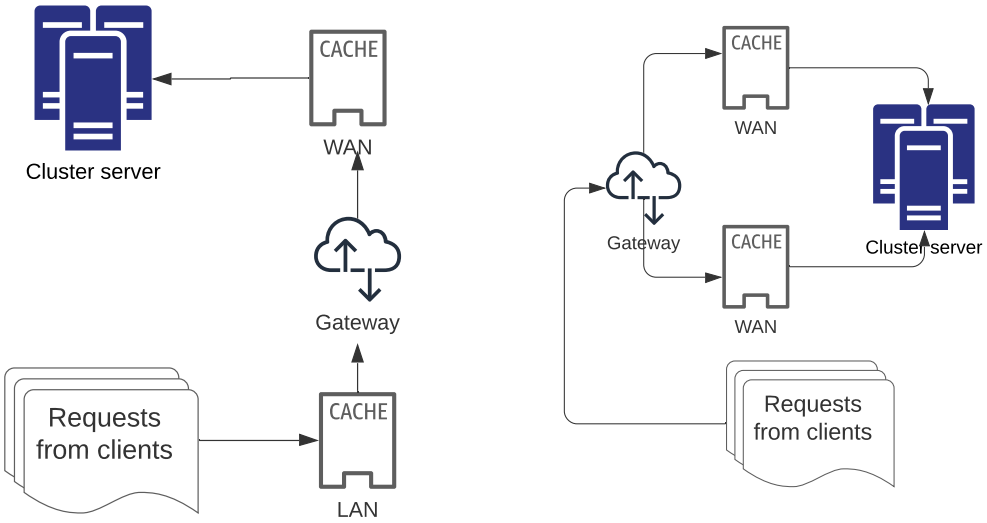
Regardless of the cache's architectural level, the intercommunication between the cache and processors constitutes a performance concern because the cache is first consulted before accessing the processor for execution, thus increasing the execution stages. However, the throughput and response time still benefit from the cache if there is a high hit rate. The use of caching in distributed systems then becomes paramount for performance improvement.

There are two main parameters that impact cache performance. On the one hand it is the cache size, which determines the amount of data that be stored, and therefore how likely is to have a hit according to popularity and requests frequency. On the other hand is the replacement policy, which refers to the criteria to select the data for deletion if space is needed for new entries. The most commonly used replacement policies are First-In-First-Out (FIFO) and Least Recently Used (LRU). This project refers to a cache running LRU replacement policy.

Cache optimizations vary according to the metric, as [15] shows. Development turns around decreasing the hit time, increasing the cache bandwidth, reducing the miss penalty, and reducing the miss rate. Due to the approach and goal of this project, the analysis in the case study focused on reducing the average access time to the cache that equals the reduction of hit time. The latter approach is mainly affected by the cache size and the cache associativity [15]. The cache size was the main interest of this project, as explained in Chapter 3.

There is an increasing trend for multi-branch organizations to share remote resources through remote caches. However, the cost of remote queries differs according to the infrastructure and cache location. Then, an optimal organization is needed when

implementing a shared cache. There are two types of architectures to implement remote cache systems: hierarchical and distributed [16]. The main difference between these architectures is where to place the cache unit. Figure 2.2 illustrates a hierarchical cache to the left and a distributed cache to the right. This project refers to a remote distributed cache.



(a) Hierarchical **(b)** Distributed

Figure 2.2: Types of architectures for remote caches.

To speed up the access time to the cache, there exists the content-based storage management. This approach, also known as Content Addressable Storage (CAS), is the most efficient technique to speed up the requests in non-rewritable repositories [11]. On the contrary to the location-based lookup, there is no need to traverse the entire memory address to get the target and retrieve the content. Instead, the lookup process relies on a hash table, also known as Action Cache (AC), to store the actions and their results as hash codes in a key-value pair fashion. The case study of this project relies on a $AC + CAS$ remote cache.

In spite of the lookup mechanism, the popularity of data items is of great concern for caching systems because of the effects that data duration and requests frequency have on cache performance performance [17]. Here, the common approach is to assume an Independent Reference Model for inter-requests time. However, the content popularity does change over the time [18], which leads to constant caches requests that consume bandwidth and takes read and write time. In the end, each cache request impacts on overall cache performance.

2.2 A systematic approach for performance evaluation

As computer systems are unique in organization and resources, any evaluation should be adjusted to their components and conditions. For example, performance evaluation in distributed systems usually cares for individual or overall throughput, latency, and utilization, leading to further analysis and experiment designs. In addition, performance evaluation aims to identify weaknesses, commonly known as bottlenecks, towards improving the configuration for the system to deliver its highest performance at the lowest cost. Hence, it is also known that the more granulated the evaluation, the more accurate the results.

Many authors have worked on techniques for performance evaluation of computer systems. The proposals apply to different systems regardless of their architecture and resources availability. For instance, Jain [19] proposed a systematic approach to conduct a performance evaluation by characterizing the system's features. Le Boudec [2] described different methodologies for performance evaluation that turn around the workload and the metrics instead. In the end, a clear system description is what leads to meaningful insights. Below is a summary of the most prominent findings to carry out a performance evaluation of computer systems, that were applied in this project.

Despite the system organization and the type of analysis to perform, there should be an objective to address. By stating a goal, it becomes easier to identify inputs, outputs, and metrics of interest. For instance, this project aims to evaluate the performance of a system composed of two main services: remote cache and remote execution cluster, in order to identify bottlenecks and points of improvement. Therefore, it can be stated that the input is the workload from the compilation requests, the output corresponds to the results, and the metrics of interest are the *throughput* and the *response time*. Both metrics are typically used in performance evaluation, the throughput measures the amount of job processed per time unit, and the response time measures the time it takes for a job to be served by the system.

The metrics intend to quantify the service delivery and compare it to the expected or desired limits. There is a trade-off between complexity and accuracy for the metrics. However, metrics applied to the entire systems also allow to get relevant insights, even if a subsystem is not working at its highest. For the metrics to be relevant, it is highly recommended to express them in function of any of the system's features as the workload or available resources.

After an overall description, more details are needed for each component. Hence, it is needed to characterize the individual services and their outcomes. Referring to the case study of this project it can be said that remote cache is mainly comanded by its size with the aim of provide storage for previous results; whereas the remote execution cluster is mainly comanded by the number of nodes in the cluster with the aim of load-balancing the workload.

System's parameters	Workload's parameters
Cache size	Number of actions
Cache replacement policy	Source size
Number of workers	Locality of requests
Network bandwidth	Arrival rate of actions

Table 2.1: Parameters that affect directly the performance of a remote cache and remote execution system.

All things considered, performance is not affected only by the workload but its configuration. For instance, parameters from Table 2.1 are considered of significant impact on performance of the case study system. The parameters are classified as system parameters and workload parameters. System parameters are related to hardware and software, while workload parameters refer to input features as the users and their demand. Evaluations and analysis can be validated through experimentation. To emulate real systems infrastrcuture and workload is challenging. However, there are multiple techniques such as simulations and small-scale configurations. Experiments are designed for data gathering, analysis, and results presentation. More details about the experimental setup used in this project is described in Section 3.5.

2.3 Techniques for computer systems modeling

Likewise performance evaluation, there exists multiple approaches for performance modeling of computer systems. The work of Lazowska [20] constitutes the foundations for most of the contemporary works. In essence, a model intends to represent the system's features and behavior as algebraic relationships. Stochastic processes are usually used to characterize the workload flow, and the occurrence of events due to their property of randomness [21].

Modeling can start in observation, if trace records are available, they might help to identify patterns such as the arrival rate of incoming jobs. As an example, Figure 2.3 depicts the density distribution of incoming workload to the remote execution cluster from the case study. It shows a fluctuation in the workload, with peaks that suggest the system is working at its highest and can be the starting point of an evaluation towards an analytical model.

Observation can also be useful to identify statistical distributions that represent the variables of a system's model by simply comparing the observed pattern of the records with the density distributions of the existent random variables (e.g., binomial, exponential, Zipf, chi). For instance, let the incoming jobs be independent one from another, and their arrival rate looks like Figure 2.3. It can be said that the input workload is a discrete random variable, x, that follows an exponential distribution. Thus, the exponential distribution is used to represent the input workload mathematically.

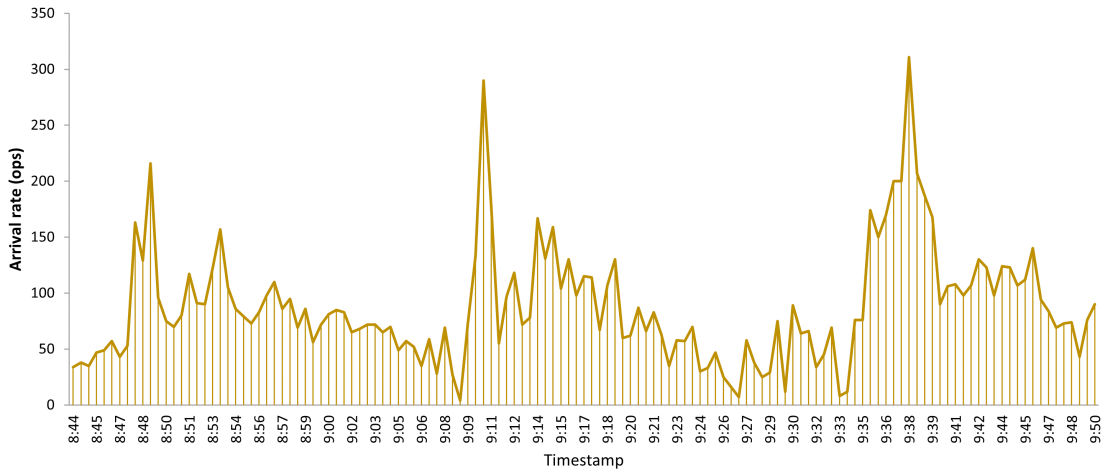


Figure 2.3: Density distribution of the arrival rate of tasks, from queueing to execution, in the remote execution cluster.

Moreover, if the time between arrivals follows an exponential distribution and are independent one fom another, then it can be also said that the arrival process is a Poisson process. The Poisson process is defined by a parameter, λ, that represents the average number of arrivals per time unit (i.e. the arrival rate). The Poisson process is widely used to represent input workloads when modeling computer systems [22].

Apart from the workload, a model characterizes the services as well. The main properties of services are the time it takes to serve the requests (a.k.a. service time), and the capacity, which is the total amount of the resource. Complex models also consider waiting time and the time it takes for the job to shift among services in interactive systems. The model proposed in this project reviews the role of waiting time in the system, but considers only the service time for computations.

Similarly to performance evaluation, it is essential for the model to use the notation of outputs as functions of inputs. It gives way for tuning tests, as well as better understanding of the system capacity. For instance, the throughput as a function of the arrival rate. There exist several mechanisms for systems representation, each with different levels of accuracy and complexity on performance projections. This project performed an analytical study based on modeling.

An analytical model can rely on stochastic models to represent its parameters, as previously exemplified. However, performance analysis is not that manageable when using stochastic models. Instead, there is a broader technique that allows making approximations to simplify the analysis [21], the queueing theory, which is introduced below.

2.3.1 Queueing Networks

Queueing network approach is a concept used to describe a system behavior by queueing theory; it represents the system as a series of queues that handle the incoming requests [20]. Figure 2.4 illustrates this concept by depicting a single-server queue network. A queueing network can also represent a system with multiple servers. Modeling a system as a queueing network allows to understand the impact and relation between incoming requests and capacity through mathematical equations derived from each component's features [23]. In the case of complex systems with multiple queues, the resulting chain makes the analysis harder by requiring to keep track of the requests through each queue [21].

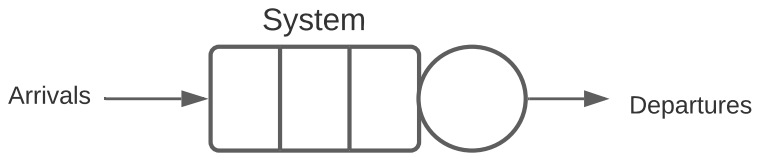


Figure 2.4: Abstract overview of a queueing model.

Fortunately, there are different queueing networks systems to represent different types of computer systems. For instance, a system in which the inputs and outputs are external is called an open system. On contrary, if the jobs feed multiple stations within the network, but are not delivered outside upon completion, it is a closed system. Of course, the assumptions and goals differ in open and closed systems. Typically, an open system assumes that the number of departures equals the number of arrivals, making one of the metrics, the throughput, known. On the other hand, a closed system does not know the departure rate but the number of jobs in the system at every time unit [22] [19].

Queueing networks are described by Kendall's notation. This standar notation has the form $A/B/c/K/L/Z$, where each character stands for a specific queue feature, as explained below.

- A: inter arrival time distribution or arrival process (e.g. Poisson).
- B: service time distribution.
- C: number of servers in the queue station.
- K: system capacity, in terms of number of jobs the queue is able to buffer.
- L: size of the source from which the jobs arrive (i.e. population size).
- Z: scheduling discipline (e.g. FCFS).

For convenience, Kendall's notation can be simplied if some of the features are assummed as their default values. For instance, if K and L are infinite, and the queues follows a First-Come-First-Served scheduling policy, the notation omits K, L and Z characters so the notation is shorter. On the other hand, A, B and c should be clearly stated since they are the statistical definitions of the queue. Thus, a multiserver system with exponential inter arrival time, exponential service time, and c servers is denoted as a $M/M/c$ queue because M states for exponential distributions [24].

A network of $M/M/c$ queues, where each queue has its arrival rate and service time, can be subject of the *product-form* solution (i.e. individual analysis of the service stations) if they are modeled and treated as *Jackson's Networks* [19]. However, in order for a queing network to apply Jackson's theorem as it stands in Equation 2.1, the following conditions must be satisfied:

1. All the servers are identical with an average service rate μ.
2. The jobs are expected to arrive to the stations from either outside the system or from other station. External arrivals are assummed to follow a Poisson process with an average arrival rate λ.
3. The served jobs are able to either leave the system or go to another station in a probabilistic routing, such that all the outgoing probabilities from a station sum 1.

Thus, the average arrival rate of each service station, in a network composed of k $M/M/c$ queues, is given by the expression below [24].

$$\lambda_k = \gamma_k + \sum_{i=1}^{K} P_{ik}\lambda_j \tag{2.1}$$

Where, γ_k equals the external arrivals to station k, and P_{ik} equals the probability of the jobs to arrive at station k from a station i. Jackson's theorem is also known as the *Flow Balance* equation, since the system is assummed to be in equilibrium. In addition, Jackson's theorem provides the probability of the overall system's state. It is, the probability that there are N_k jobs in the k_{th} station, but the study of this probability goes beyond the scope of this project.

In the model, the incoming jobs equal the tasks to be processed in the system, whereas the system's capacity equals the system's resources to process the tasks (i.e. CPU, RAM, network bandwidth, storage space). A multi-server queueing model gives way to represent components that serve with different resources. An accurate representation of the workload is needed to avoid misleading results about the system's behavior. When modeling, the workload is mainly characterized by its arrival rate, expressed in *ops* (jobs per second). Although it is impossible to replay a real trace as the workload for modeling, observation gives way to probabilistic representations of patterns instead.

Service parametrization is another essential procedure for modeling because it de-

fines the characteristics of the queue. A service station is mainly defined by its service time, and its capacity. The service time is the time that the station spend serving a task. From the service time it is possible to get the service rate, which is the number of jobs served by time unit. Furthermore, the service capacity is the amount of tasks that the station is able to buffer before, during or after the service. Usually, the models consider an infinite buffer capacity for the queues in the systes, even though it might lead to inaccurate results for capacity planning and performance evaluations.

Depending on the approach for the analysis, the parameters can be described using stochastic models, measuring the real system, or approximating according to the observed in a real scenario. In the end, the system's outputs are quantitative measures of how the service performed to the workload, called metrics. Typically, throughput and response time are the metrics used for performance evaluation, especially in open systems. However, individual metrics can be defined per service stations if the services are different. Therefore, one of the major concerns when stating an analytical model is to find the relationship between the workload, the system operation, and the output. Some fundamental laws already state some of such relations and are presented below.

2.3.2 Fundamental Laws

Despite the fact that deterministic models eases the stochastic process description, there are some relations that do not require detailed descriptions about the system to accomplish a performance analysis. The primary benefit of using operational laws is that real system measurements or simulations can verify the results [19] [20].

Let Figure 2.5 illustrates a system modeled as a single server queueing network

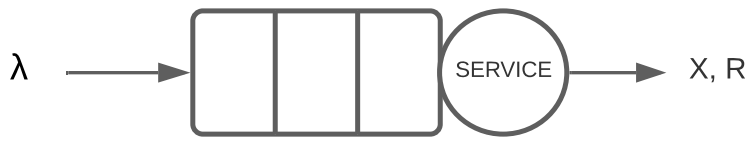


Figure 2.5: Parameters of a single server queueing network model.

The quantities of interest and notation from the illustrated system are introduced in Table 2.2. Where, λ and μ are random variables with exponential distributions, and *ops* stands for operations per second. Also, operations equals to jobs in this project.

Notations	Description	Units
λ	System's arrival rate of jobs	ops
N	Number of jobs in the system	jobs
T	Time a job spends in the system	seconds
μ	Service rate	seconds
X	System's throughput	ops
R	System's response time	seconds

Table 2.2: Notation used for queueing network models.

2.3.2.1 Little's Law

Motivated by the need to connect the system's performance with the input, *Little's Law* is the first and most important law for performance analysis. It relates the average number of jobs in the system with the average time a job spends in the system and the arrival rate. Little's Law is applicable to both open and closed systems. As for illustration, consider an open system where the job flow is expected to be balanced (i.e., the number of departures is equal to the number of arrivals), thus $X = \lambda$. Equation 2.2 express Little's Law to apply on open systems, where $E[N]$ is the average number of jobs in the system at a certain time t, and $E[T]$ is the average response time experience by the tasks that were served during a certain time t.

$$E[N] = \lambda \times E[T] \tag{2.2}$$

In this project, the Little's Law is used to model the response time by increasing the number of jobs in the system, after tuning the system's features such as number of servers and service time.

2.3.2.2 Utilization Law

A system's performance is not only result of the amount of workload the system receives, but how much the resources are used. Hence, *Utilization Law* in Equation 2.3 relates the arrival and service rates such that the utilization reflects if the system has the capacity to handle the workload.

$$\rho = \frac{\lambda}{\mu} \tag{2.3}$$

The previous relations are, indeed, a straightforward beginning for performance evaluation in a single-server queueing network. However, to identify performance weaknesses in a multi-server system, it is required to find the congestion from a series of services in a system. The *Bottleneck Analysis* approach, described in [22] in the context of queueing networks, aims to give insights about overloaded stations under the premise that the utilization, μ, should be lower or equal to 1.

In case a station has more than one server to distribute and serve the workload, then

Equation 2.3 is modified such the utilization is divided by the number of servers, c, of the station. Thus resulting in Equation 2.4.

$$\rho = \frac{\lambda}{\mu \times c} \tag{2.4}$$

Then, a service station with $\rho > 1$ is an overloaded service and a potential bottleneck since its performance limit has been reached. It is expected that any improvement to overcome bottlenecks, even in just one service station, would have significant impact on the overall system's performance.

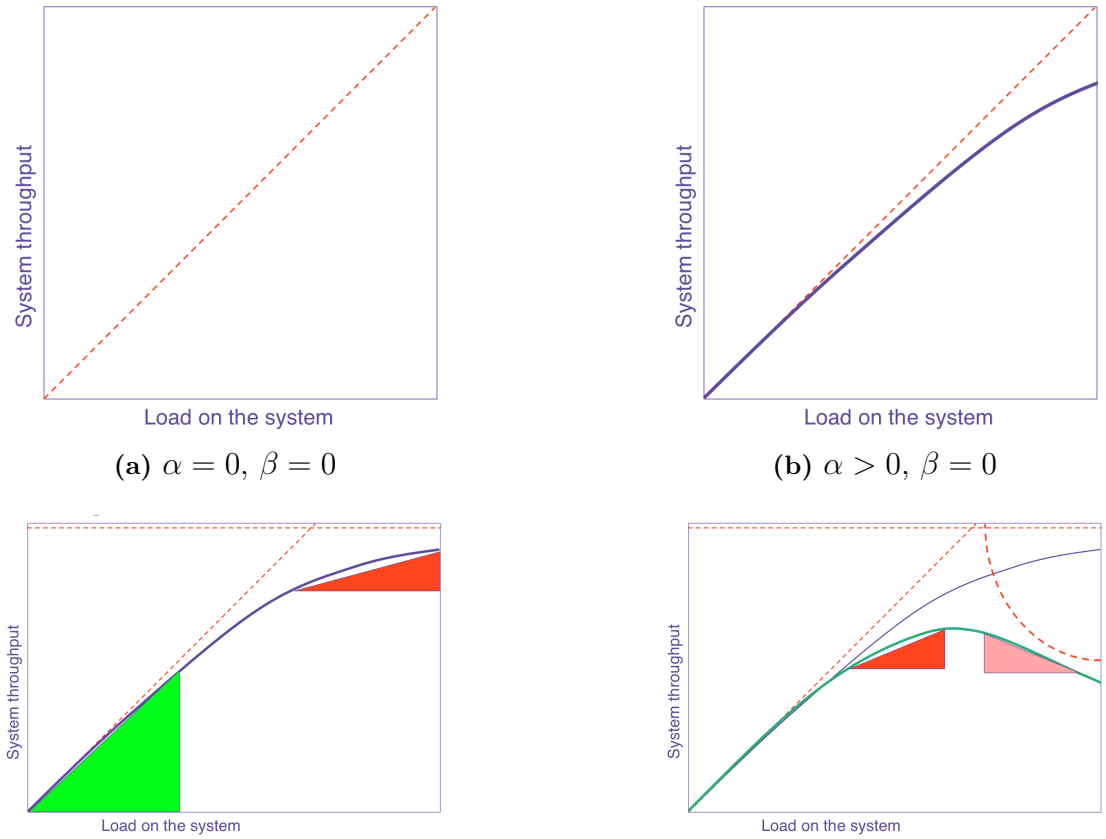
2.3.2.3 Universal Scalability Law

Finally, a nonlinear model developed by Gunther in 1993 [25] allows to predict the system's behavior by combining the well known Amdahl's Law and Gustafson's Law, in a relationship called *Universal Scalability Law* (USL). The expression derives from the aim of quantify performance scalability in parallel processing [14]. On contrary to queueing theory, this model does not need for service measurements as input. Then, the throughput, X, and the workload, N, are related in Equation 2.5 alongside three coefficients, α, β and γ, which represent contention, coherence, and concurrency, respectively.

$$X(N) = \frac{\gamma \times N}{1 + \alpha(N-1) + \beta N(N-1)} \tag{2.5}$$

The author demonstrates that USL reduces to Amdahl's Law when $\beta = 0$ and $\gamma = 1$, it is, ideal parallelism and data consistency. In this project, USL is used to find the optimal concurrency level and predict the throughput of the best configuration. Figure 2.6 depicts the scaling effects that can be identified or predicted using USL [14]. Here, scenario (a) corresponds to linear, the ideal, scalability with no contention and total consistency among the nodes. Then, scenarios (b) and (c) represents the effects of contention, it is, the cost of sharing resources among multiple nodes. High contention can lead to bound system's throughput which, in turn, alongside inconsistency can lead to retrograde throughput by bounding the load as well. Scenario (d) is most likely to show up when there is a high exchange activity among the nodes, as in the case study configuration [14].

As observed, scalability is more sensitive to contention effects, such that, the higher the contention the more bounded the scalability is with respect to the throughput and load. This study concluded the type of scalability the system is expected to perform according to the experimental configuration. Currently, applying USL to production data is of research interest. Further details about the system and adaptation of modeling techniques are explained in Chapter 3.



(a) $\alpha = 0,\ \beta = 0$ **(b)** $\alpha > 0,\ \beta = 0$

(c) $\alpha \gg 0,\ \beta = 0$ **(d)** $\alpha \gg 0,\ \beta > 0$

Figure 2.6: Scaling effects than can be illustrated by the application of USL.

2.4 Related Work

Among the multiple methods for modeling computer systems, queueing networks have been used effectively due to their versatility and simplicity when analyzing system performance. While this method may compromise accuracy, some authors have developed good approximations. The following review inspired the approach used in this project.

Most of the works in performance modeling of computer systems, using queueing networks, study Web systems. Similar to this project, the author of [26] obtained measurements from load tests, for performance evaluation, to subsequently use queueing networks for modeling and analysis of distributed environments. In the end, the methodology resulted suitable to evaluate different systems architectures under a variety of workloads. The response time was a metric determined by the model. The study concluded, among others, that adding new devices (horizontal scalability) is not always the solution for performance improvement.

Since analytical modeling is considered an effective evaluation approach by implementing queueing networks, multiple researches have extended the simple models to

more complex ones by considering heterogeneity from the workload. For instance, Palmer et al. in [5], used a queueing network consisting of J multi-server services to provide an efficient approximation method to model systems with complex flow dynamics. This approach intends to understand the effects of such flow over the dependency between overall demand and system capacity, and how resources may be managed in light of them with heterogenous costumers.

Likewise, Jiang et al. developed a hierarchical model to understand the behaviour of Cloud-edge data centers with heterogenous workloads. The system was described as a hierarchical queueing model, where response time was analyzed as a function of the arrival rate. The accuracy of the model proposed in [27] was validated with analytical-numerical solutions and simulations. A similar approach was presented by Chang et al. [28], who proposed a hierarchical stochastic model for performance analysis under heterogeneous workload in scenarios where each costumer job may require a different number of virtual CPUs. The model quantifies the impact of arrival rate, buffer (cache) size, and number of CPUs, on performance.

However, it is not always possible to know the system organization and monitor its behaviour. Therefore, estimating the model parameters becomes a challenge. Awad et al. [29] developed a method to estimate service demands for open, close, and multi-class queueing networks. In the end, the method aims to measure the resopnse time for different workload intensity. A closed or open QN with multiple server queues can be approximated into a single server QN using this approximation.

Other researchs point to decomposition as the most effective way to understand system parametrization. Kuehn [30] published one of the earliest and most influential studies on queueing networks by decomposing them individually and proposing an approximate method. In the study, N stations with single server queues are distributed over an open network. The interarrival times and service times are exponentially distributed. Research revealed that generality in the arrival process is more influential on accuracy than service process. Furthermore, the study recommends the approximation to close networks for better results because of dependencies. According to the author, the single-server stations can also be replaced by multi-server stations.

In addition to general performance evaluation models, there are also specific models for clusters and caching systems. For instance, Vilaplana et al. [31] based their work on queueing theory and open Jackson's network to develop a model for cloud architecture design that guarantees a certain level of performance. Since the model combines $M/M/1$ and $M/M/c$ queues, it is a valid reference for this project. Moreover, the model evaluates the performance through the response time and determines the bottlenecks to correct the proper affecting parameter. Likewise, Ahmed et al. [32] suggested a model for performance evaluation by estimating the throughput of a cluster or the expected service time of the tasks. This work also detects bottlenecks by using a multi-class Jackson network.

As for the caching systems, there are also performance models that have been developed using queueing networks. Even though most of them characterize proxy caches in web services, the foundations are a significant starting point for the aims of this project. For instance, Bérczes et al. [33] updated the model from [34] to analyze the impact of arrival rate requests in the performance of a proxy cache server by considering the arrival rate as a Poisson process and the service times as random variable following an exponential distribution. The numerical experiments revealed that the response time increases with the arrival rate of requests, while more insights were obtained from tuning the cache hit rate probability. Another widely studied approach is to evaluate the performance of multitier applications, where at least one of the tiers acts as a caching system. Hence, it is possible to introduce interaction among the queues through probabilistic routing, as [35]. This scheme allows assessing the natural effect of cache hits on the overall system's performance.

The nature of computer systems and their distinct features do not give way to generic frameworks for performance modeling. Albeit, previous research sets precedence for meaningful comparisons of methodologies and results. This project applies the most used approach (i.e., queueing networks) for performance modeling, but it differs in the system architecture and parameters. Particularly the impact of cached requests on the overall response time of system. Also, the complementary approaches to identify scalability bounds from the results obtained in the evaluation. Finally, the experimental setup using Bazel implementation has not been an object of study but sets a configuration that provides uniqueness to the project.

3
Methodology

This chapter aims to describe a general-purpose methodology for performance evaluation, which in turn is illustrated on the actual system. The system modeling applies queueing theory, whereas the performance analysis was conducted through experimentation and prediction. The latter requires to apply the universal scalability law exposed in Section 2.3.2 in the previous Chapter. The methodology followed in this project is summarized in three main stages: data gathering from real system measurements, describing the system as a queueing network with mathematical relations of performance metrics, and model validation using experimental scenarios.

The content of this chapter proceeds as follows, Section 3.1 describes the system and its operation as a function of the remote cache and remote execution units. Then, Section 3.2 shows meaningful measurements from the real system that allows characterizing the system towards an operational analysis, aiming to identify bottlenecks. Next hypothesis and assumptions are stated in Section 3.4. Finally, the simulation environment is described in Section 3.5.

3.1 System Description

This project refers to the case study of the industrial partner, precisely, the infrastructure used for software development. In the case study, the clients are the developers who build, compile, and test different software projects. Therefore, the infrastructure provides for remote execution and remote cache services to clients through the implementation of Bazel. This project focuses on the core features of that specific infrastructure. An overview of the system's operation is illustrated in Figure 3.1 to point out the paths that incoming jobs might take to be served by the system.
The system then operates as follows. The source code stipulates the environmental variables, commands, arguments, inputs, and outputs for compilation, from which actions are identified according to a dependency graph built by Bazel. Bazel avoids executing actions if the inputs, arguments, environmental variables, and commands correspond to previous executions. The code is then separated into actions according to the building dependencies. Each action specifies all the inputs, including the command to be executed.

Results from previous executions are stored in the Conten Addressable Storage (CAS) unit, and mapped in the Action Cache (AC) as a key-value entry. The AC

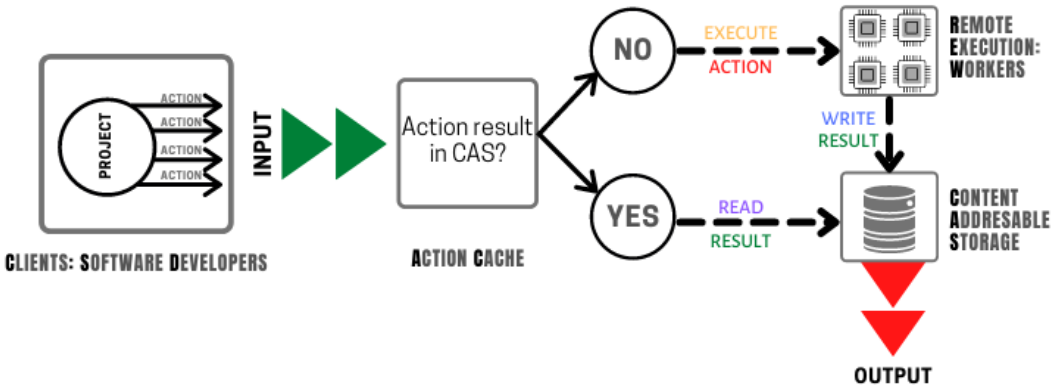


Figure 3.1: Overview of the operation of the case study system.

module is responsible for managing a hash table with the correspondence between the actions and the results of the actions [11]. A successful lookup in the AC gives way to retrieve the action results from the CAS. Otherwise, the remote execution unit executes and puts the result in the CAS for subsequent requests.

In the production system, the cache is an array of two disks, 6TB size each. The addressing space is divided by half among the two disks, which distribute the IO operations. Derived from Chapter 2, it is known that a high hit rate from the cache guarantees a faster output delivery. However, it depends on the system capacity (i.e., the cache size in storage units), so the larger the cache, the higher number of results to store for subsequent requests. Nevertheless, storage is not infinite, and so the number of items needs to be replaced from time to time when the cache is full. Here, the replacement policy comes into action. Since the impact of the replacement policy is beyond the scope of the project, a Least Recently Used (LRU) replacement policy is assumed from now on.

On the other hand, the remote execution unit is studied as a cluster. The cluster capacity is devoted to the number of nodes in the cluster and the supported level of parallelism. Moreover, the cluster consists of 19 virtual machines, each running a Docker container capable of executing the actions using a maximum of 32 threads per container. The threads are also known as the buildbarn workers for Bazel. Despite the number of servers that each container can handle, there is a physical limitation in the host VMs. There are only eight cores available per VM to serve the threads. In other words, under a heavy workload condition, there is a maximum of 608 tasks (i.e., threads), but only 152 can be executed in parallel.

3.2 Measuring the Real System

As mentioned previously, observation is a practical starting point to identify a system's behavior. Tools for system monitoring primarily intend to measure the resources consumption at each component to compute essential metrics such as throughput and response time. The measurements are obtained by querying the

operating system, and the metrics result from relationships and operations. For instance, the response time results from the difference between the time a task leaves the system and the time it entered, therefore obtaining how long it takes to be processed. Moreover, modern monitoring tools also provide interactive dashboards with plots to follow and understand the trends through time.

Two efficient and widely used open-source tools for systems monitoring are Grafana and Prometheus. Prometheus is a monitoring and alerting toolkit that collects and stores metrics as time-series data in a multi-dimensional data model. Prometheus also supports a query language for the data to be exported. On the other hand is Grafana, a visualization solution that retrieves information from a data source, as Prometheus, to plot as convenient. Grafana also provides a collection of shared dashboards that fits different requirements and data sources [36].

The system of the case study uses Prometheus and Grafana for monitoring of the remote cache and remote execution cluster. This project requires the metrics listed in Table 3.1 for the operational analysis, described in the following Section.

Service	Metrics	Units
Remote Cache	Operation rate by operation (*GET, PUT*)	ops
	Duration by operation (GET, PUT)	seconds
	Operation rate by GRPC status (OK, Not Found)	ops
Remote Execution Cluster	Operation rate	ops
	Duration by stage(Queueing, Executing)	seconds
	Operation count by stage (Queued, Executed, Completed)	operations

Table 3.1: Metrics available from Prometheus and Grafana monitoring in the production system of the case study.

Hence, the metrics corresponding to the remote cache are retrieved from the Action Cache (AC) and the Content Addressable Storage (CAS), where

- Operation rate by operation is the number of read and write operations per time unit reported by the CAS. This metric is used to compute and μ of the remote cache.
- Duration by operation correspond to the average time it takes for the read and write operations. This metric is reported by the CAS and is used to compute R of the remote cache.
- Operation rate by GRPC status corresponds to the number of operations per time unit reported by the AC as OK or Nor Found when the lookup is successful or not, respectively. This metric is used to compute λ and X of the remote cache.

As for the remote execution cluster, the metrics are reported by the Scheduler and the Build Executor units, which corresponds to the Frontend and the workers, respectively. Metrics from the remote execution are also mean for the operational

analysis, where

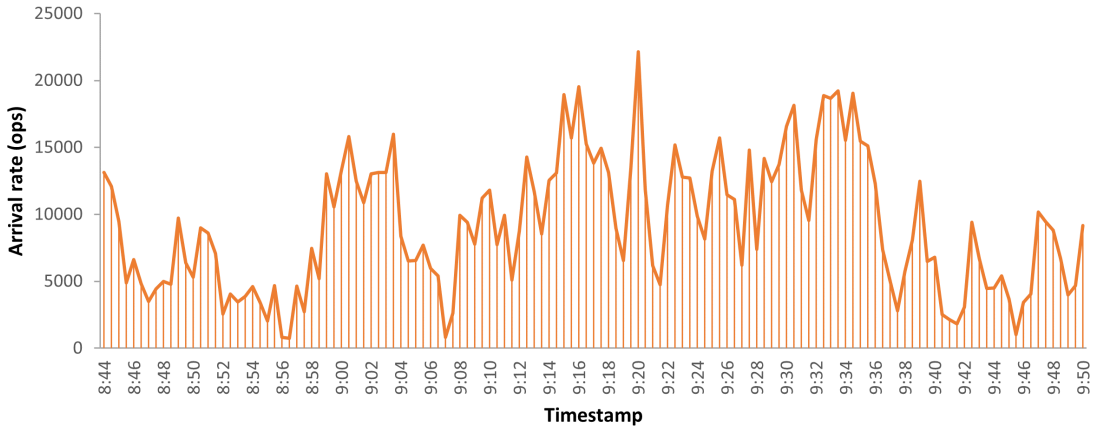
- Operation rate is reported by the Scheduler as the number of operations per second assigned to the workers. Since the number of active workers equals the number of threads, this metric corresponds to the number of active threads per second.
- Duration by stage is reported by the workers, as the time it takes for the tasks to wait in the queue and be executed. It is measured in seconds and R in the cluster.
- Operation count by stage is reported by the Scheduler as the number of operations in the queue, being executed, and mark as complete. λ and X are obtained from this metric.

Analyzing the metrics from the monitoring system is the first step to understand the system's behavior and recognize performance anomalies. For instance, if a long task duration is identified by observing a dashboard, another dashboard can be analyzed within the same time series, looking for a pattern such as a high operation count or high arrival rate. If the irregularity coincides, then it can be suggested that the factor impacts the metric. As an illustration, Figure 1.2 (b) shows long responses time from the remote execution cluster around 08H55; at the same time, Figure 2.3 displays a peak on arrival rates to the remote execution cluster. Therefore, an assumption can be that the response time increases with the arrival rate. Then, the analysis can turn around the confirmation of that proposition.

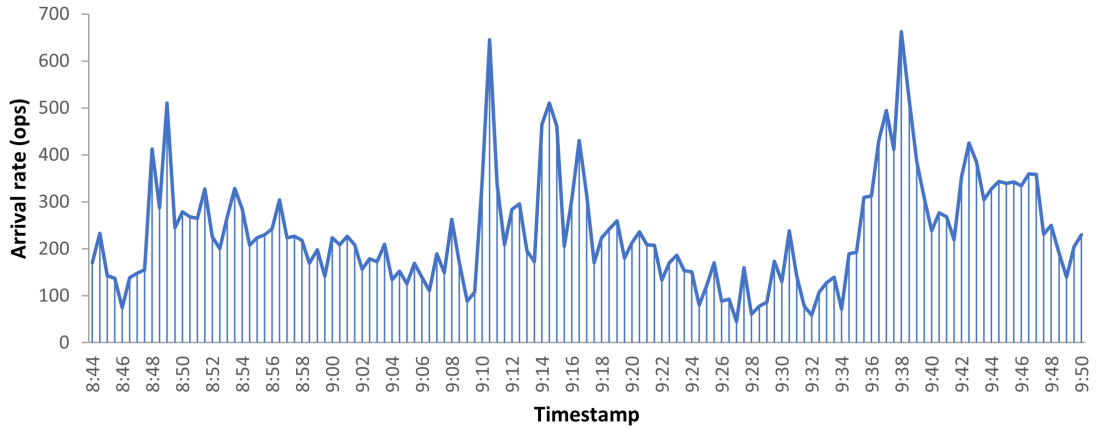
Particularly, in this project, observing data from monitoring gives way to understand the entire system by, for example, compare the arrival rate of jobs to the cache with the remote execution cluster. It can be observed in Figure 3.2 that the number of operations per second arriving to the cache is significantly higher than the number of operations per second arriving to remote execution cluster, within the same time interval of 66 seconds. Therefore, it can be inferred that the remtoe cache is more demanded than the remote execution cluster.

Besides, the remote cache and the remote execution cluster can be analyzed in isolation. As for the remote execution cluster, the scheduler manages the incoming actions as queued, executed, and completed. Queueing status refers to the jobs that are scheduled and wait to be served. Then the executing status refers to the jobs that the workers are already executing. Finally, completion refers to the jobs that are ready for leaving the system. Even though this project focuses on the execution stage, the queueing stage is also important to observe because it can relate to a long response time. Illustrated with Figure 3.3 is the density distribution of job duration by stage in the cluster. It can be seen that the most populated stage is the executing stage. The sporadic population on the queueing stage leads to infer that long-stay times result from non-available workers.

Another significant measure from the system is the number of workers that are busy during the execution stage. From the monitoring, it is known that the total



(a) Remote cache.

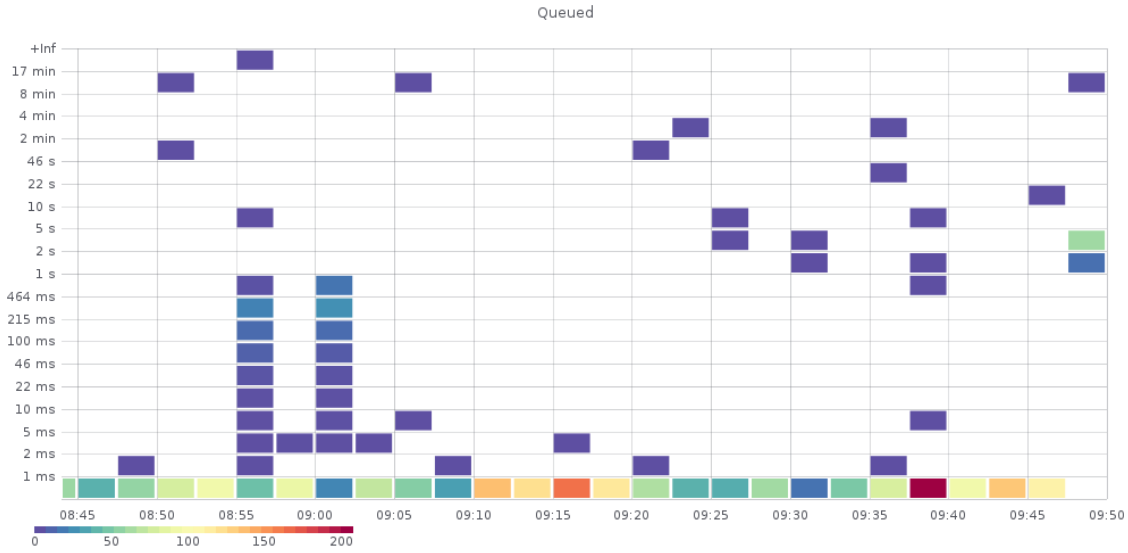


(b) Remote execution cluster.

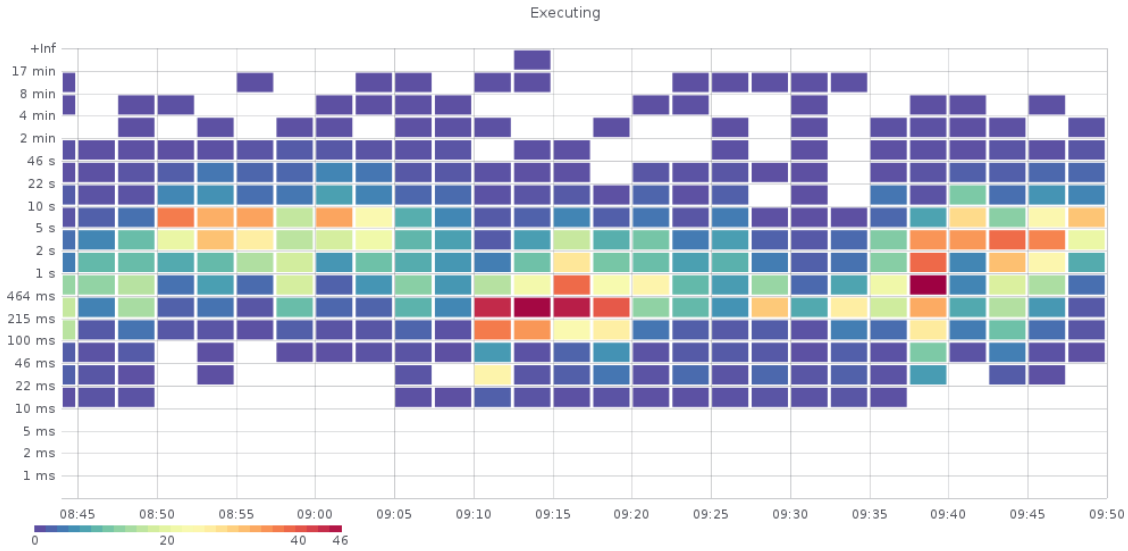
Figure 3.2: Probability distribution of arrival rate of jobs to the remote cache and the remote execution cluster.

number of executing workers corresponds to the total number of tasks that are being executed concurrently. For example, figure 3.4 shows the density of busy workers during eight hours.
Below are summarized the most relevant insights obtained from observations that have been illustrated thus far in this Section:

1. There are more results retrieved from the cache than executed by the cluster. Also, the cache is the fastest service in the system. Then, a model is needed to evaluate that increasing the cache capacity would benefit the overall response time of the system.
2. The highest queueing times coincide with the highest executing times. However, the number of busy workers is not the same, nor the maximum, in the most demanding time intervals. Then, a performance evaluation of different configurations could determine the best arrangement to exploit the cluster's capacity.



(a) Queueing Stage



(b) Executing Stage

Figure 3.3: Job duration distribution by stage in the remote execution cluster.

So then, monitoring data allows to evaluate the system's performance in terms of the stated metrics: Response Time (**R**) and Throughput (**X**). Precisely, the response time from advertised for the cache, R_c, corresponds to the average time it takes for *GET* operations to be completed. In addition, the cache throughput, X_c, corresponds to the number of successful operations per second that the cache delivers. PUT requests are not considered in the workload since the number of GET requests is considerably greater and only GET requests leave the system back to the user.

Figure 3.5 depicts R_c and its trend by arrival rate of GET and PUT requests.

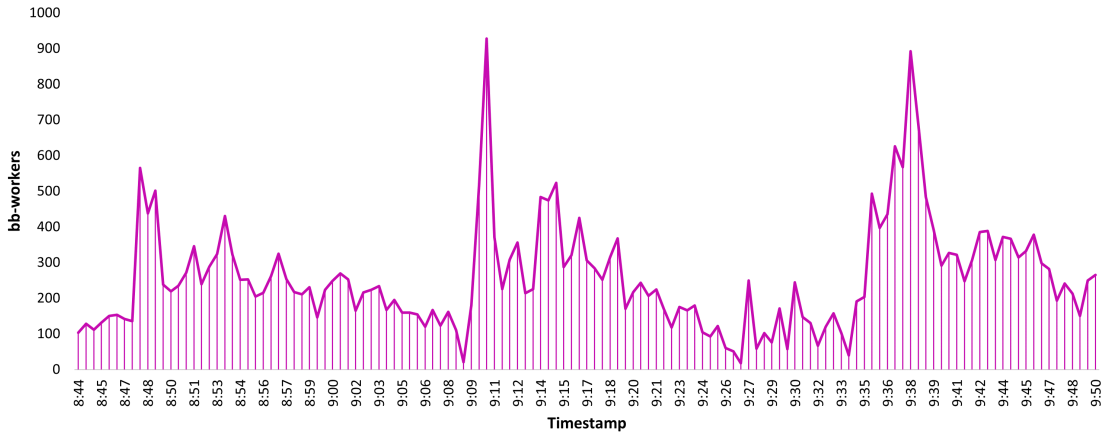


Figure 3.4: Density distribution of busy workers under regular workload execution in the remote execution cluster.

As observed, the response time of PUT requests is not affected by increasing the arrival rate, as the response time of GET requests does. This is because the larger the buffer, the longer it takes for the cache to perform the lookup.

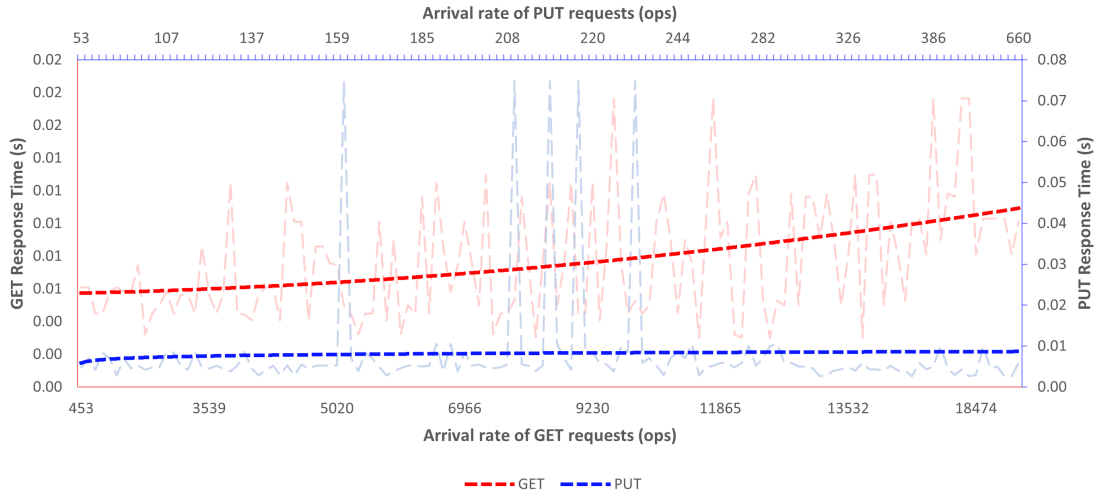


Figure 3.5: Measured response time by arrival rate of GET and PUT requests in the remote cache.

Likewise, throughput was measured concerning the arrival rate for both GET and PUT requests. As illustrated in Figure 3.6, PUT requests are served almost at the same rate as their arrivals. However, serving GET requests is affected as the number of arrivals increases and does not reach the expected rate.

The same metrics were obtained from the remote execution cluster. Here, the *scheduler* unit advertises the arrival rate, the number of jobs by stage (i.e., queueing and executing), and the residence time of jobs by stage as well. Because of the stages, the response time of this subsystem corresponds to the sum of the queueing time and

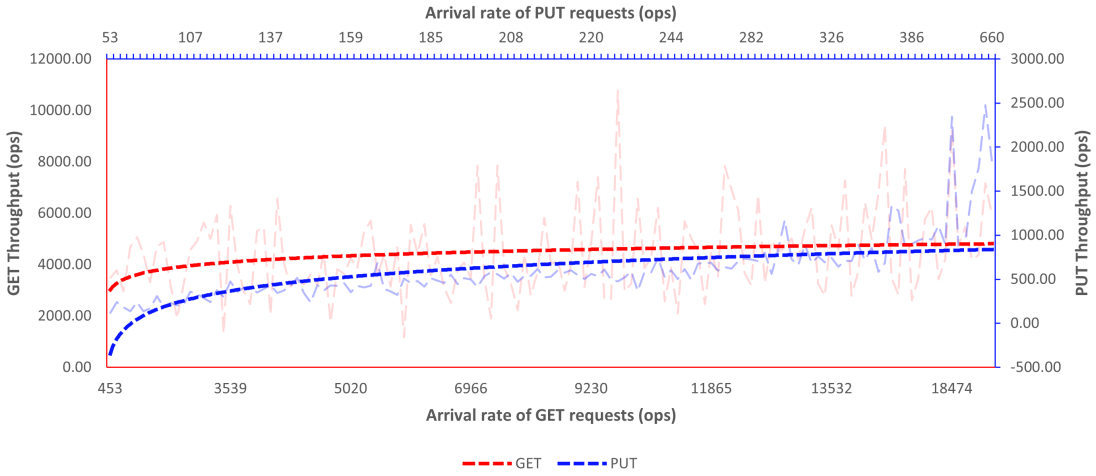


Figure 3.6: Measured throughput by the arrival rate of GET and PUT requests in the remote cache.

execution time, which in turn is shown in Figure 3.7. Throughput is also depicted as a function of the number of jobs in the system in Figure 3.8.

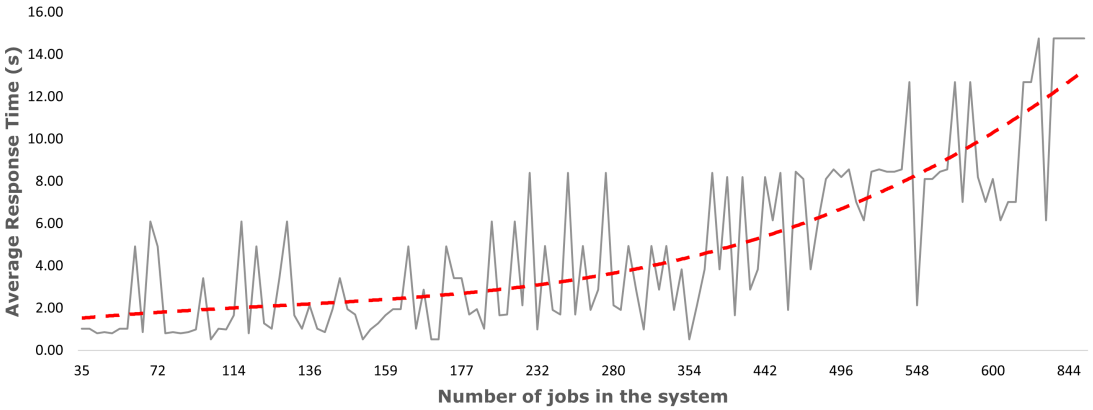


Figure 3.7: Measured response time by the number of jobs in the remote execution cluster.

In brief, from the magnitude of the response times from the cache and the remote execution cluster, the overall response time is mainly affected by the time it takes for the workers to execute the jobs. Likewise, the overall system's performance can benefit from improving the cache. For that reason, the following sections aim to analyze the best strategies to improve the cache and the execution time in the cluster.

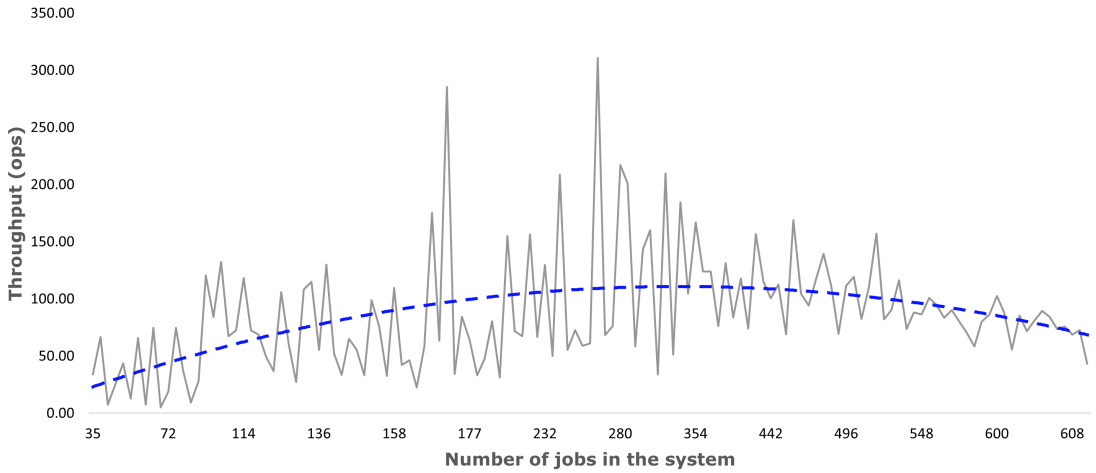


Figure 3.8: Measured throughput by the number of jobs in the remote execution cluster.

3.3 Operational Analysis

The following analysis aims to ease the understanding of the existing real system's performance by applying the fundamental laws reviewed in Section 2.3.2. The values used in the computations correspond to real values from the actual system. Notations from Table 2.2 are used for the quantities.

Even though there are several meaningful results from a performance analysis, this work focuses on detecting bottlenecks. Then, the study starts with the identification of the most demanded service between the remote cache and remote execution. For that purpose, Equation 2.4 is applied to determine the utilization of each service. Recall that each each parameter from the Equation 2.4 is described twice with different index, c for the cache and w for the remote execution cluster. Then, the following parameters are used:

- $\lambda_{\mathbf{c}}$ is the arrival rate of requests to the cache, equals the GRPC messages returning status *OK* in the Action Cache (AC).

- $\mu_{\mathbf{c}}$ is the service rate of the cache, obtained from the operation rate of GET and PUT operations in the Content Addressable Storage (CAS).

- $\mathbf{c_c}$ is the number of servers acting as cache during the observation period. This number is fixed to two since there are two cache servers in the system object of study.

- $\lambda_{\mathbf{w}}$ is the arrival rate of requests to the remote execution cluster, equals to the operation rate that pass from the *Non-existed* status to *Queued* status, advertised by the scheduler.

- $\mu_{\mathbf{w}}$ is the service rate of the remote execution cluster, obtained from the operation rate advertised by the buildbarn workers.

- $\mathbf{c_w}$ is the number of servers that build the cluster. However, the number of busy workers varies according to the workload because of the load distribution. Therefore, the value is obtained from the count advertised by the scheduler.

The values were gathered from an observation period of 66 minutes when the system operated under a wide range of workloads. Firstly, the average number of jobs in the cache, N_c, was computed with Equation 2.2, where R_c is the response time of the cache. Then, the cache utilization was computed by applying Equation 2.4 to obtain ρ_c. Figure 3.9 shows the impact of arrival rate and the number of jobs in the system on cache utilization. It can be observed the overutilization of the cache once it overcomes the threshold of 1.

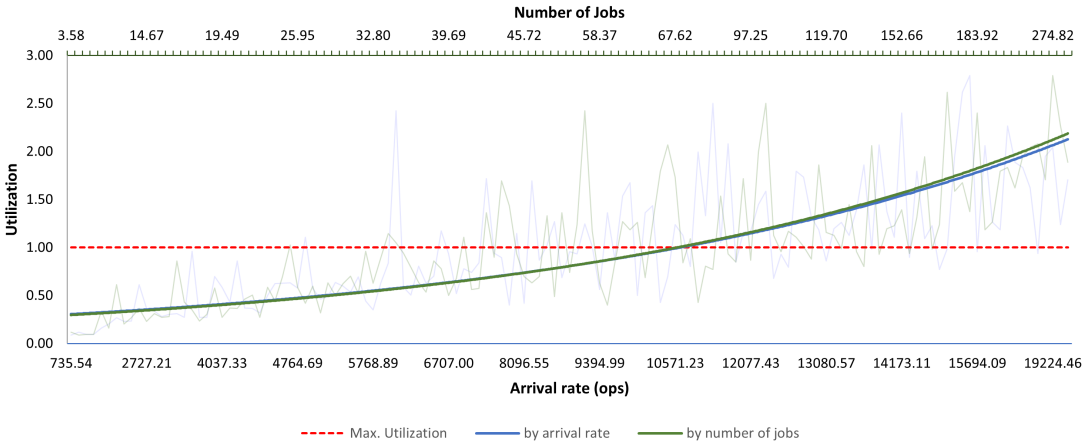


Figure 3.9: Measured cache utilization by arrival rate and number of jobs.

As for the remote execution cluster, the number of jobs in the subsystem is not required to compute since it is already provided by the production monitoring. Then, Equation 2.4 is applied to determine the utilization of the cluster under the given workload conditions. Figure 3.10 depicts the cluster utilization, ρ_w, with respect to the number of jobs in the cluster. Note that the number of jobs correspond to the sum of queued jobs and executing jobs.

Furthermore, by monitoring the CPU utilization in the cluster, it can be seen in Figure 3.11 that the total number of cores is far from been used, even with the maximum number of threads assigned to a job. Recall that the configuration of the case study is of 152 total cores and 608 total threads.

In brief, measurements from the existing system suggest that the remote execution cluster is not over-utilized. Unlike the remote cache, the cluster utilization does not reach the threshold of 1, even with an increasing number of jobs in the system. However, the average response time of the remote execution cluster is higher than the

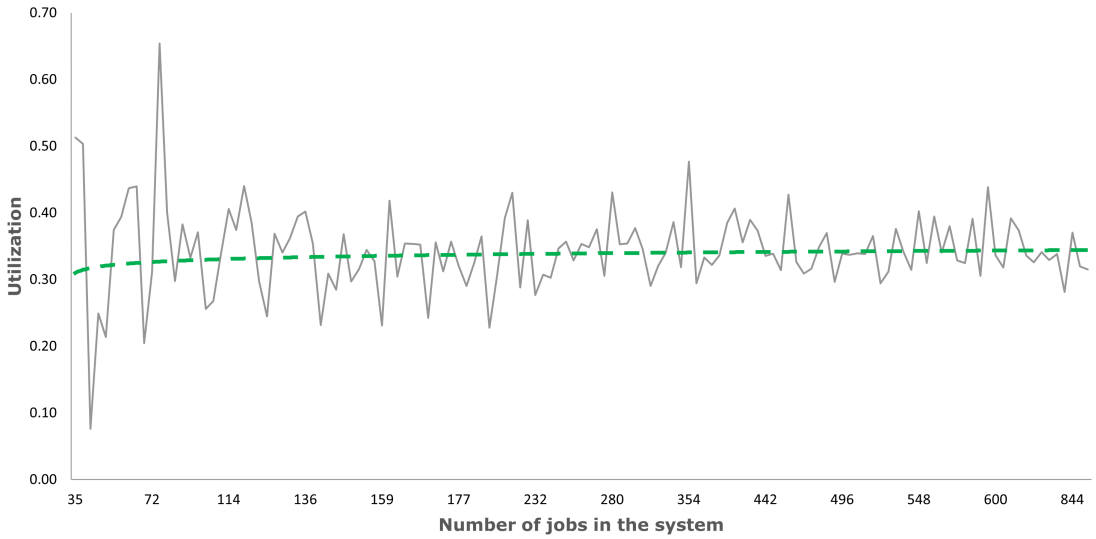


Figure 3.10: Measured cluster utilization by number of jobs in the remote execution cluster.

Figure 3.11: Monitored number of cores used by number of jobs during execution stage in the remote execution cluster.

remote cache and increases sharply, leading to suspect hardware scalability limits. Thus, from the operational analysis results, the system needs to strengthen the cache and speed up the cluster. A model is therefore developed to test the options towards the assumptions mentioned above.

3.4 Analytical Model

Following the performance modeling approach described in Section 2.3, the case study is modeled as a multi-server queueing network. It is treated as an open Jack-

son Network model with independent queue stations corresponding to the remote cache and execution cluster. Since each station needs a different analysis, and Jackson's theorem allows each queue to be studied in isolation, then unique parameters and capacities came up with the definition of each queue. The proposed queueing network model is depicted in Figure 3.12.

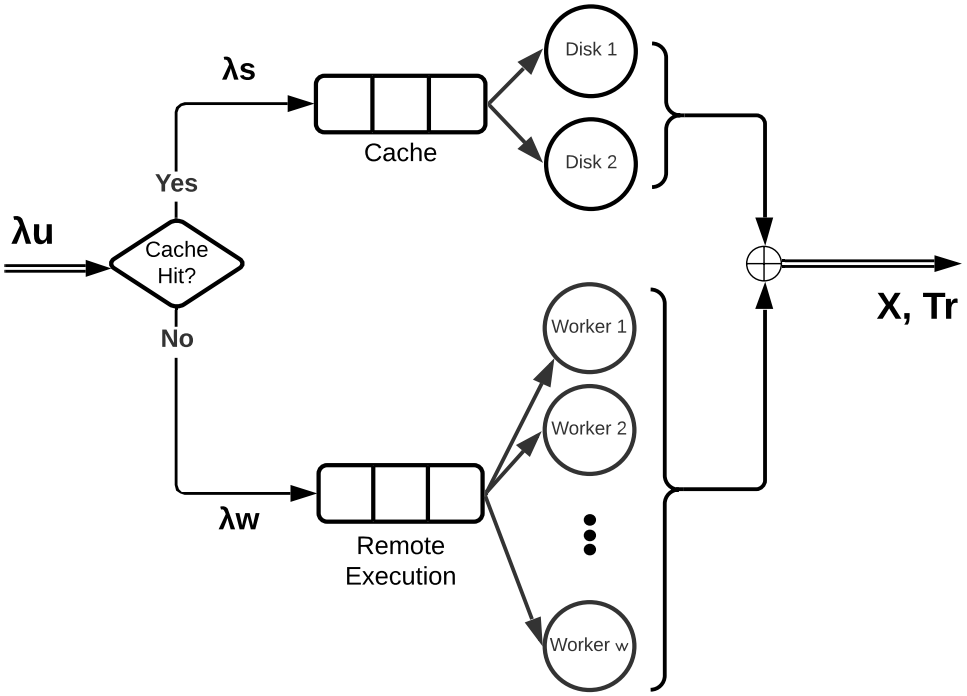


Figure 3.12: Queueing network representation of the system.

In the end, the model delivered individual insights that were relevant to the overall performance overview.

3.4.1 Hypothesis and Assumptions

The assumptions for the model turn around the case study system. The model intends to be as representative and straightforward as possible. The scheduler is not part of the model as a unit, but its function is considered in the Remote Execution queue subsystem. Thus, it is assumed that the scheduler follows a FIFO discipline to dispatch the requests to the workers. In the whole system, it is also considered that the number of incoming requests is independent of the system capacity. Finally, every non-execution overhead, such as network transmission latency, is included in the service time. It depends only on the size of the requests and is considered non-congested.

As for the cache, the number of disks and their sizes are the leading performance concerns. However, it is important to remark that the content-addressable storage (CAS) implements a least recently used (LRU) replacement policy. Since the cache

is modeled as a multi-server queue, the servers are assumed of the same size. Therefore, the model starts evaluating two servers, as the case study. Like the cluster, the cache performance is related to data transfer, and then the network bandwidth is a vital factor. However, it is assumed that the server's location and the network bandwidth optimally support the data transfer rate.

On the other hand, the remote execution unit is assumed to be a cluster of ω workers, each capable of running up to ϕ threads to process the incoming requests. The threads are independent, and there are no idle threads when the scheduler is full of requests. Queueing and execution times are considered for the computation of the response time. There is also assumed a processor-sharing policy for load balancing among the workers and the threads. What is more, job-thread affinity is ignored for simplicity, which implies ignoring locality in each worker.

3.4.2 Performance Modeling

An analytical model starts with the characterization of the system. Refer to notation from Table 2.2 and Figure 3.12 for the following description. The input workload corresponds to the number of arrival jobs, and is measured in operations per second [*ops*]. The input is characterized by its arrival rate, assummed as a random variable with exponential distribution that follows a Poisson process with parameter λ. The overall arrival rate of jobs is then divided in λ_c and λ_w such that $\lambda = P_c \times \lambda_c + (1 - P_c) \times \lambda_w$, where P_c is the probability of the request's result to be on the cache.

The system's outputs are the actions successfully served by both, the cache and the remote execution. Therefore, the metrics of interest in this project, measured over the system's outputs, are the throughput $X(\lambda)$ and the response time R, such that:

- $X(\lambda)$ counts the number of completed jobs by arrival rate, and
- R is the time it takes for a job to be served, from entering to exiting.

Recall that each job can be served by either the cache or the remote execution cluster. Hence, $X(\lambda)$ is measured in operations per second [*ops*], and R is measured in seconds [s].

By considering to model a system in equilibrium, (i.e. $\lambda = X$), then Little's Law is adapted to get the system's throughput, $X(\lambda)$ as a function of the number of jobs in the system, N, and the time a job spends in the system, R. Thus Equation 2.2 can also be expressed as Equation 3.1 below.

$$X(\lambda) = \frac{N}{R} \tag{3.1}$$

Regarding the response time R, it is conceptually defined as the addition of waiting and service times spent by a job in every unit (i.e., cache and remote execution).

Nevertheless, waiting time is not considered for the cache, and its service time corresponds to the duration of GET() operation in the Content Addressable Storage. As for the remote execution, both the waiting and service times are considered as their definitions. Thus, the overall response time R is computed by Equation 3.2, where R_c and R_w are the average response times reported by the cache and the remote execution cluster, respectively; λ_c and λ_w correspond to the arrival rates to the cache and remote execution cluster, respectively; and P_c is the probability for the jobs to go to be served by the cache instead of the cluster.

$$T_r = \lambda_c R_c P_c + \lambda_w R_w (1 - P_c) \tag{3.2}$$

The model is based on a $M/M/c$ queue where, the input workload is an arrival burst of jobs that follow a Poisson process, thus the interarrival time of jobs is exponentially distributed. Furthermore, the service time has a general distribution, and the number of servers, c, corresponds the number of workers, ω, in the cluster.

The application of Little's Law is regarded as a gold standard tool for performance modeling. Therefore, Equation 3.1 evaluates the remote execution unit under the following definitions: As Q is the number of jobs in the system, it equals to the number of cache misses out of the total incoming jobs. From the real system it is known number of jobs served by remote executors. Thus, the expected value of λ_w equals to Q_w as the mean number of jobs served by the remote execution unit.

3.5 Performance Prediction

This section describes the experimental scenario used to try different configurations of remote execution services, in order to predict their performance and select the best approach. Recall that the metrics used in the project are the response time and the throughput, thus measured in the experimental setup.

The laboratory environment intends to be a system representation of the case study of this project as illustrated in Figure 3.13. Here, both the cache and remote execution units are replicated from production configuration. It was mounted on a server running XCP-ng hypervisor to host the Linux-based virtual machines that act as *frontend*, *backend*, and *workers*.

Due to the CPUs availability from the server, two out of the three workers were configured with 1 virtual CPU each. In a later configuration, one worker is powered off for the second worker to have two virtual CPUs. A third worker was added in the final stage to create a new experimental scenario for comparison. The third worker had 4 CPUs. Configurations from Table 3.2 were used with the same load each, to measure and compare the throuhput and response time.

Since the cluster has three levels of resources: number of nodes, number of cores, and number of threads (i.e. concurrency level), then concurrency was chosen as the scalability factor because it is part of the Bazel configuration and it affects the total

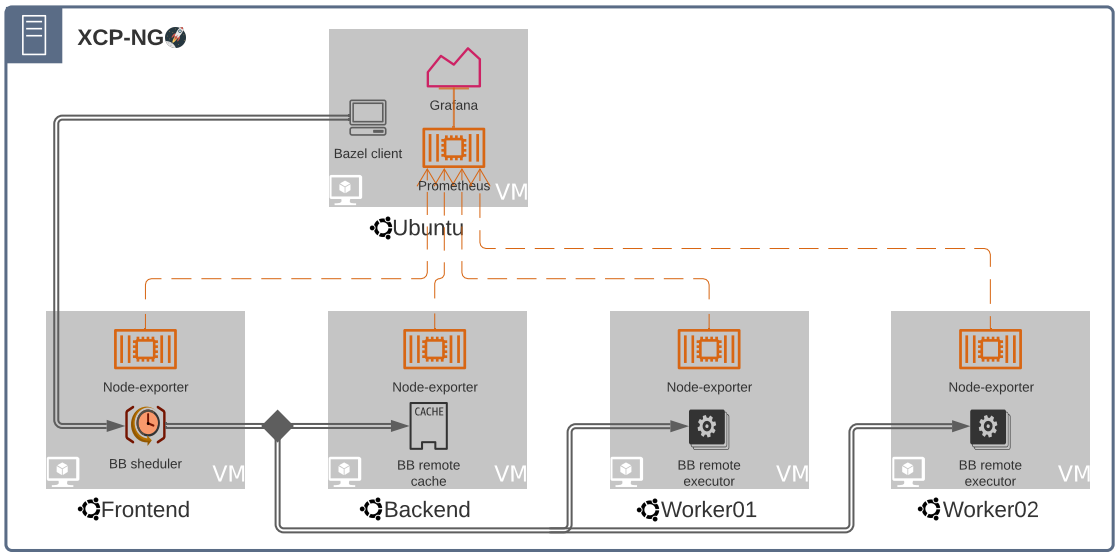


Figure 3.13: Overview of the experimental setup used to evaluate performance in different configurations.

Workers	Concurrency
1x1CPU	4,8,16,32
2x1CPU	4,8,16,32
1x2CPU	4,8,16,32
1x4CPU	4,8,16,32

Table 3.2: Cluster configurations to measure response time and throughput.

number of workers the scheduler sees as available for parallel execution. Therefore, a cluster of one worker configured with a concurrency of four equals to four threads; a cluster of two workers configured with a concurrency of four equals to eight threads. It is, the number of buildbarn workers equals the number of nodes by the concurrency level.

As for the workload, the `bazel test` command was used to compile one of the most demanded projects from production thus replicating intensive workload in the cluster. The command used the following Bazel options to specify the number of concurrent workers allowed to be used in parallel and the amount of workload, respectively.

- −−jobs: to limit the number of concurrent workers. It is, the maximum number of threads allowed to run in parallel.

- −−runs_per_test: to specify the number of times the test run. It simulates the compilation workload required by users.

The number of jobs per test run in each configuration were 100, 200, 400, 500, and 1000, and only successfully completed builds were considered. Refer to Appendix A for the datasheet of experimental configurations and results. Moreover, the use

of remote cache was avoided in every test to only evaluate the performance of remote execution cluster. Also, the services were restarted after completion of each test to reduce the effect of local caches on compilations. Monitoring of the experimental lab was performed using Prometheus and Grafana, as the production system.

Finally, USL function in R was used to forecast the scalability of the experimental cluster as a function of concurrency level. The functions reproduces the model and returns the coefficients, α, β and γ, to understand the conditions under which the system's performance is operating [37]. Before using the *usl* function, it is required to measure the throughput under controlled conditions, included the normalized measured of $X(1)$, where 1 is the concurrency level. At least a dozen of measurements are needed for the model to predict the scalability limit. Here, the evaluated system was the experimental cluster of three nodes, a total of six cores, and a concurrency level ranging from 1 to 12. Appendix B shows the code used in R and its output. The latter is explained in the next Chapter.

4

Results and Discussion

This chapter summarizes the results from the operational analysis, the analytical model, and the performance prediction, such that the most demanded service, the best configuration, and the scalability bounds, can be illustrated as a result of each operation, respectively.

4.1 Operational Analysis Results

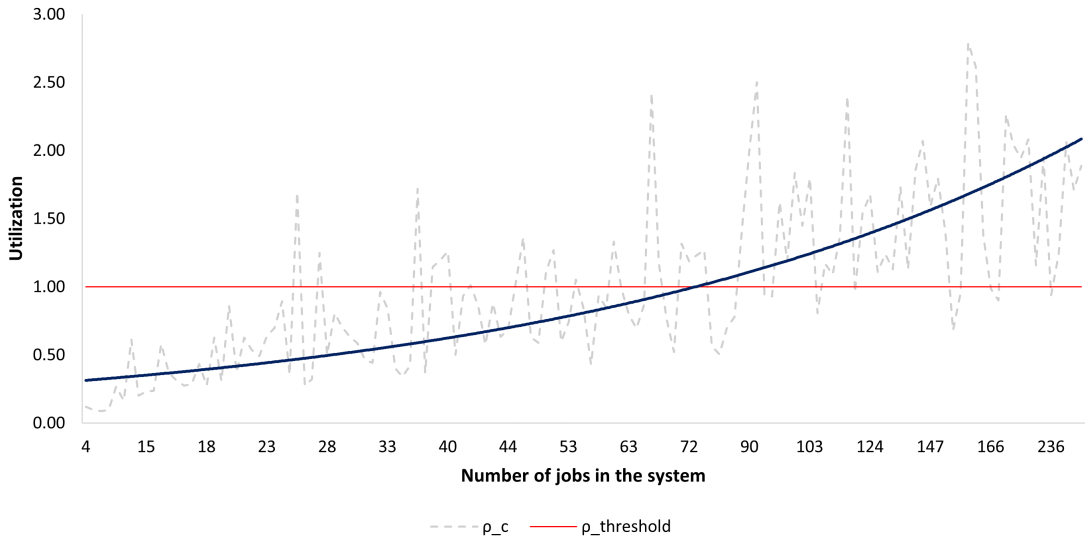
As the previous chapter stated, the aim of the operational analysis was to identify the most demanded service and therefore a system's bottleneck. By applying Equation 2.4 and parametrizing the model as Section 3.3 describes, it turned out that the cache is performing as a bottleneck because its utilization overcomes the threshold of 1 as the number of jobs in the system increases. Figure 4.1 depicts the utilization trend of both cache and remote execution cluster, by the number of jobs in the respective system.

It is important to note that, even though the cache is over utilized, it still has the lowest service time, which made the cache the perfect candidate to apply the analytical model in order to estimate performance improvements from different configurations.

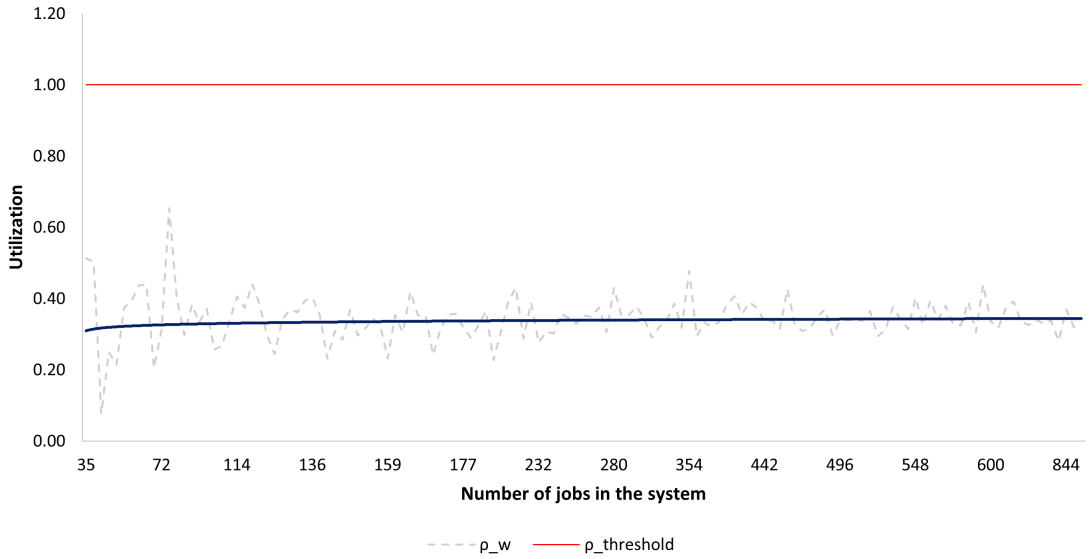
4.2 Analytical Model Results

Improving the cache can come from increasing the disk size of each cache server, or increasing the number of cache servers. It is, either vertical or horizontal scalability. The aim of applying the analytical model by parametrizing the cache, was to find the lowest response time and the highest throughput. Figures 4.2 and 4.3 show the advantages of increasing the number of caches by applying Equation 3.2 and trying different cache probabilities, versus increasing the cache capacity by reducing the service time in the cache queue subsystem.

It can be observed that, the lowest response time is obtained by increasing the amount of work sent to the cache. Instead, increasing the cache size makes the cache taking longer to perform a lookup because of the big memory page to traverse. Thus, the cache performance benefits from horizontal scalability solutions (i.e., adding servers in the queueing model). Figure 4.4 illustrates the impact of increasing the number of servers in the cache queue system, where the response time decreased by half if, for instance, doubling the number of servers from $k = 2$ to $k = 6$ in the cache



(a) Cache utilization by number of jobs in the system



(b) Cluster utilization by number of jobs in the system

Figure 4.1: Computed utilization in the cache and the cluster by number of jobs in the system by application of Utilization Law.

utilization. Note that $k = 1$ is not plotted because the initial configuration of the cache system is $k = 2$; therefore, a downgrade is not an option from the previous analysis and results.

As a result, using a minimum of four cache servers keep the cache utilization under the threshold of 1, for the workload taken from the case study.

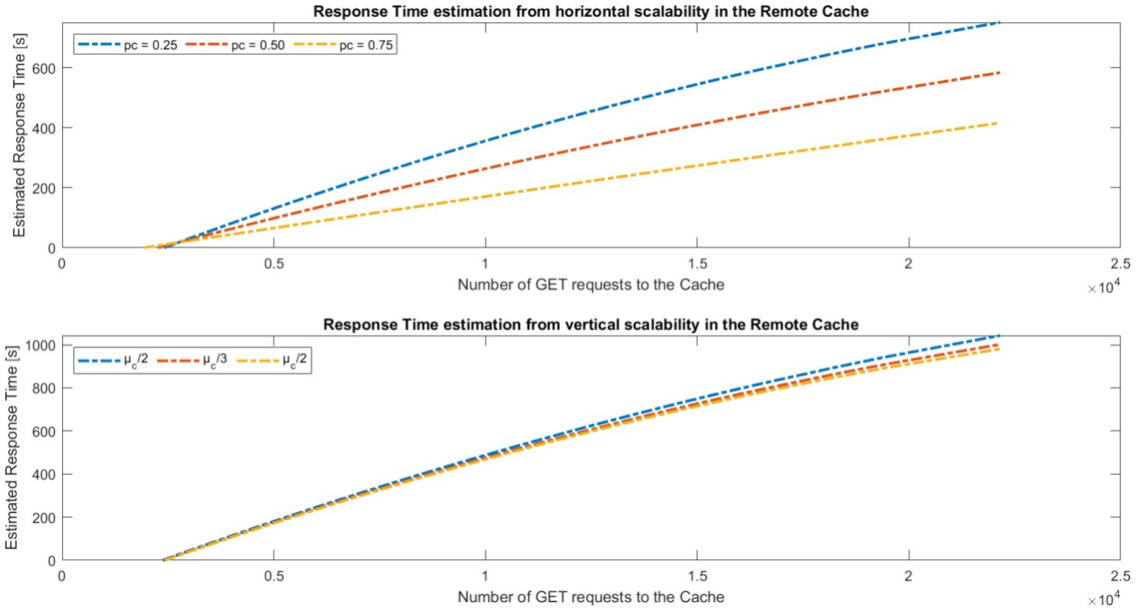


Figure 4.2: Estimated response time from horizontal and vertical scalability improvements to the cache.

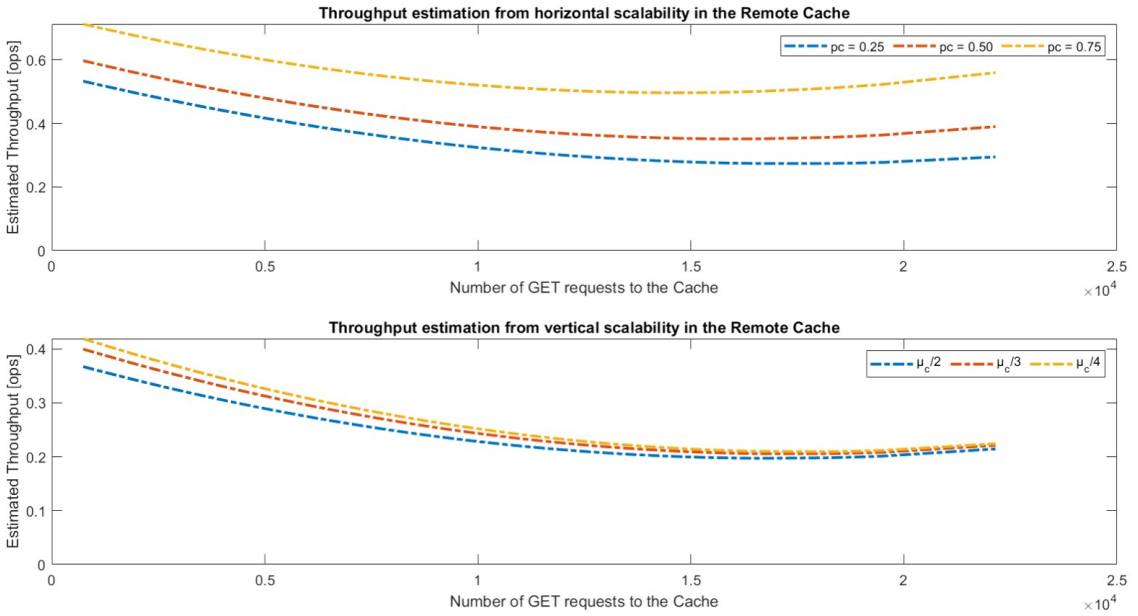


Figure 4.3: Estimated throughput from horizontal and vertical scalability improvements to the cache.

4.3 Performance Prediction Results

On contrry to the cache, the remote execution cluster was mainly evaluated by using the experimental setup and configurations from Table 3.2, due to available hardware to perform the tests. Figure 4.5 shows the measured response time and throughput, respectively, for each proposed configuration. Recall that these configurations allowed to asses vertical versus horizontal scalability effects, such that, increasing the

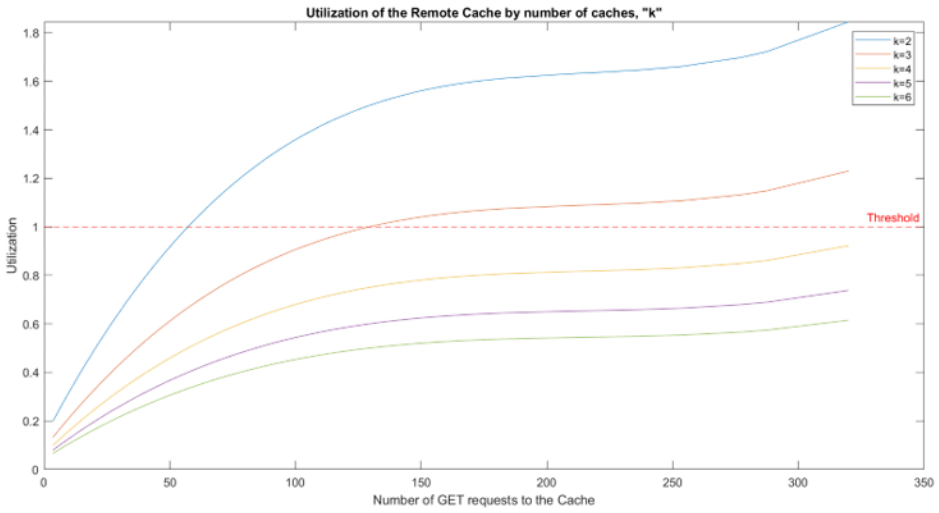


Figure 4.4: Estimated impact of the number of cache servers on the cache utilization by the number of GET requests.

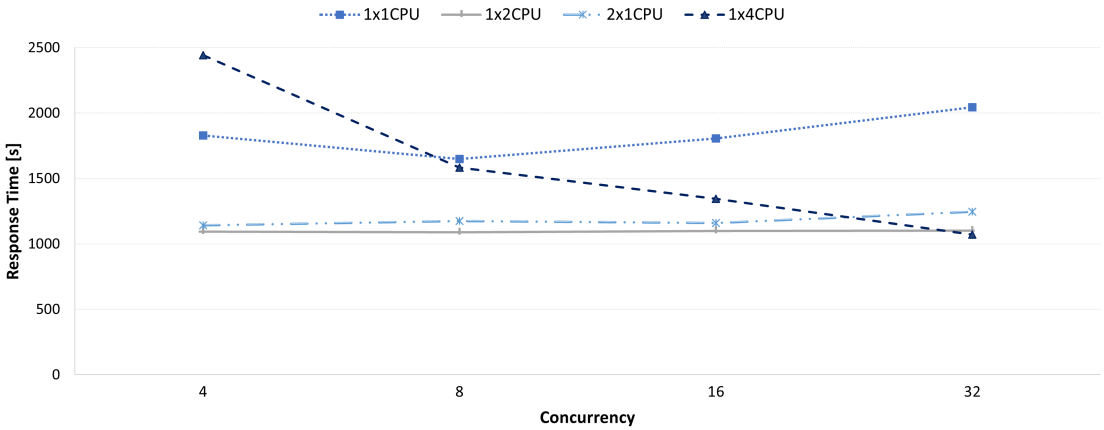
number of nodes, from one to two, represents horizontal scalability, whereas increasing the number of CPUs, from one to four, represents vertical scalability. Moreover, the concurrency level varies, such that the cluster performance was evaluated as long as the number of threads increased.

From the assessment, it can be noted that the 1x4CPUs configuration performed better when concurrency increases, compared to the 2x1CPU configuration. In other words, a higher level of concurrency is better handled by a higher number of CPUs than a higher number of nodes. Hence, better performance means the combination of highest throughput and lowest response time. Thus, vertical scalability performs better than horizontal scalability for the simulation scenario of this case study. It is important to have in mind, that the evaluations were limited by the available resources and it could not be possible to assess more configurations.

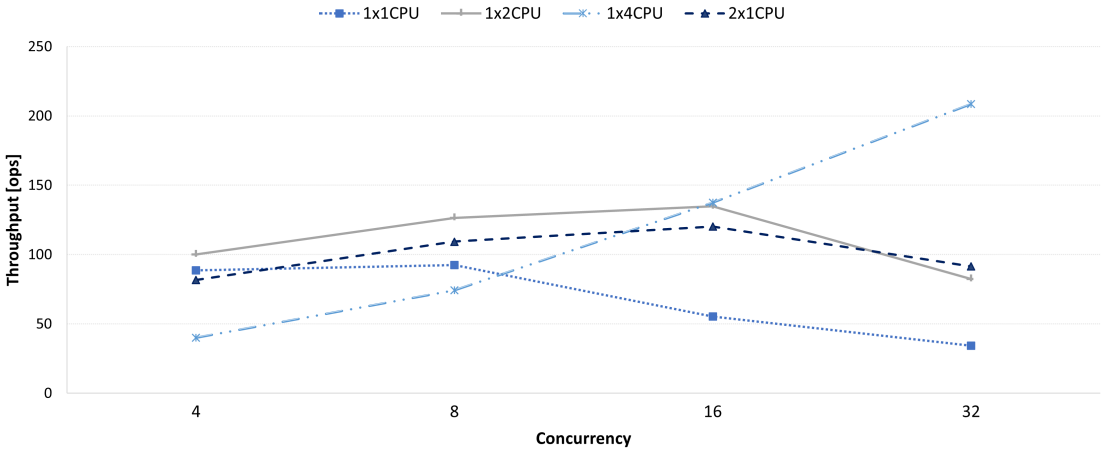
Also, the results illustrate that increasing the concurrency level within a cluster of nodes with single processor, impacts the response time and the throughput negatively. This expected behaviour responds to race conditions between the threads.

Another important insight from the evaluation comes from comparing the performance of 1x2CPU configuration versus the 2x1CPU configuration. Even though there is no significant difference that lead to conclusions, it brings to discussion the concern about synchronization among multiple nodes in a cluster. Here, it can be argued that the communication network of the system is considerable slower than the internal communication network between processors, which in the end lead to increasing latency when passing messages.

Finally, the Universal Scalability Law was applied over the previous results, specifically, over the best configuration to determine its scalability bounds. The value



(a) Measured response time by concurrency level, per cluster configuration.



(b) Measured throughput by concurrency level, per cluster configuration.

Figure 4.5: Measured response time and throughput by concurrency, for each cluster configuration arranged in the experimental setup.

pair of measured throughput by concurrency level, obtained from the 1x4CPUs configuration, were taken as the input for the *USL* model. The model is available as an open-source package for R, which applies Equation 2.5 to provide good approximations about the performance scalability trend in the assessed system. The only drawback of using the *USL* package is the need of, at least, half dozen of measured values for better accuracy on predictions. Figure 4.6 illustrates the measured and the modeled throughput, according the the USL model. Appendix B shows the code and the complete result from the USL simulation.

As a result, the model computes the limit throughput, according to the Amdahl's asymptote, as well as the peak throughput and the optimal throughput. Furthermore, the model calculate the coefficients, α, β, and γ, from which scaling effects can be deducted, as those depicted in Figure 2.6. Therefore, for the following dataset:

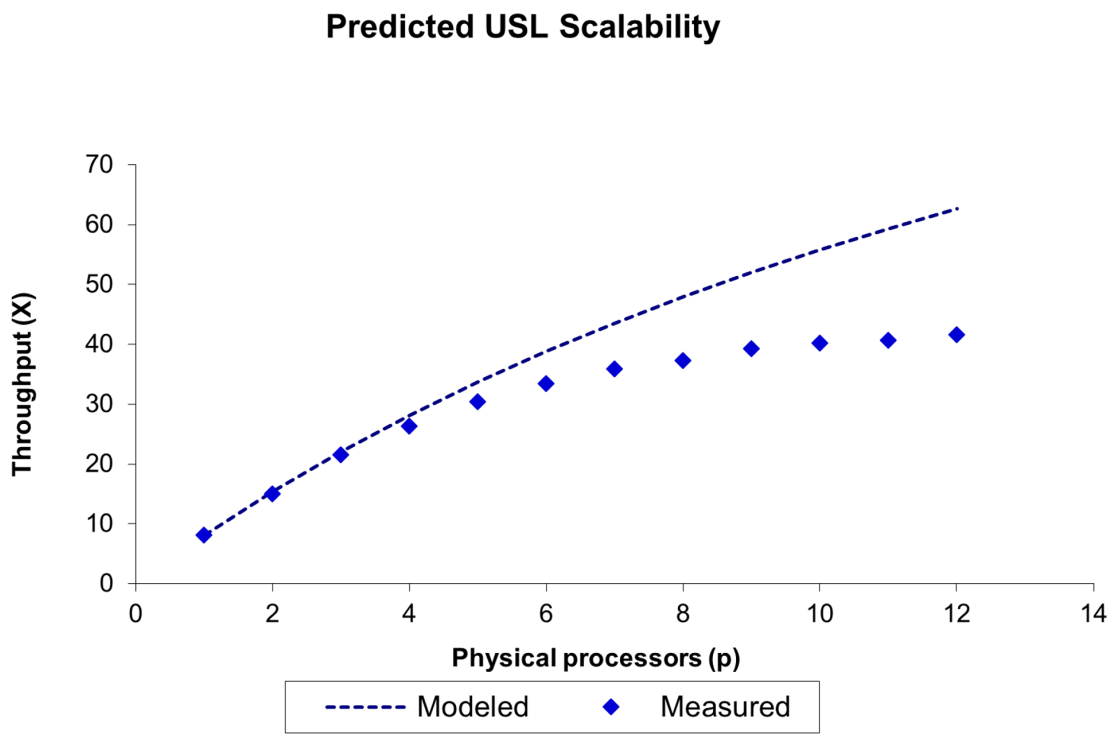


Figure 4.6: Predicted scalability according to USL model.

$$concurrency = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)$$
$$throughput = (8.09, 15.05, 21.53, 26.36, 30.41, 33.45, 35.88, 37.23, 39.27, 40.20, 40.61, 41.63)$$

The model estimate the following coefficients:
$\alpha = 0.0680652$
$\beta = 0.0049927$
$\gamma = 0.0049927$

And the following scalability bounds:
Limit throughput = 121.7
Peak throughput = 41.54 at concurrency 13.66
Optimal throughput = 41.46 at concurrency 14.69

From the results exposed above, $\beta > 0$ reveals that the system follows a retrograde throughput, which can be a consequence of multithreading. Likewise, the experimental setup performs better at a concurrency of 14.69, which does not deliver the highest throughput, but a tradeoff between performance and resources usage.

5

Conclusions

This project addressed the performance evaluation, modeling, and analysis of a particular architecture proposed by the industrial partner as a case study. The particularity of the target system was the combination of two essential techniques for high performance in distributed systems: remote caching and remote execution clustering. Moreover, the following methods were combined and used in different stages of the study: queueing networks, operational analysis (a.k.a. bottleneck analysis), and universal scalability law. In the end, the study delivered the expected results needed to answer the proposed hypothesis.

From the performance evaluation and applying the fundamental laws, it could be determined that the remote cache was an overutilized service under the reported workload. However, the operational analysis showed that the cache was also the fastest and most efficient unit in the system. Therefore, a model was used to verify that upgrading the cache represents a noticeable response time reduction. Here, it was observed that cache benefited more from horizontal than vertical scalability.

By upgrading the cache, it is important to differentiate the number of caches and the cache size. The number of caches equals the number of servers in the queueing model, whereas the cache size is the reduction of the service time. Therefore, due to the existence of an Action Cache (AC), increasing the storage capacity by cache disk would increase the lookup time and the overall response time. Instead, since the AC performs the first filter by memory address, then the optimal solution is to increase the number of servers acting as caches for the lookup time to be reduced in each one. It is important to note that, due to the particular operation of the cache, as a buffer, the results are not as accurate as expected because the replacement policy was not considered a factor in the model.

On the other hand, as for the remote execution cluster, even though the evaluation did not show any evidence of performance degradation by the arrival rate or the number of jobs in the system, the cluster was emulated by an experimental scenario aiming to find the best configuration as a tradeoff between the number of nodes in the cluster and the concurrency level. In the end, it turned out that the cluster performance benefits from vertical scalability. It is, more CPUs per node in the cluster. However, it was expected a scalability bound because of its capacity to handle contention, concurrency and coherence withing a multithreaded configuration. Therefore, USL was applied to the records giving way to estimate the optimal concurrency level, based on the measurements of the best configuration (1x4CPUs).

All in all, queueing networks technique allows to characterize a system and tune its parameters to find the best combination from the proposed enhancements. As for simplicity and accuracy, queueing networks are suitable to represent multiple systems and their interactions, as well as being completemented with multiple techniques to get better insights about performance. Vertical and horizontal scalablity don't benefit the systems in the same way, that is one of the reasons why a need for performance evaluation is needed, eventually, to estimate the capacity bounds of the system.

Bibliography

[1] K.-s. J. Hielscher, *Measurement-Based Modeling of Distributed Systems*. PhD thesis, Universität Erlangen-Nürnberg, 2008.

[2] J.-Y. Le Boudec, *Performance Evaluation of Computer and Communication Systems*. 2.3 ed., 2015.

[3] H. Ben-Ammar, Y. Hadjadj-Aoul, G. Rubino, and S. Ait-Chellouche, "On the performance analysis of distributed caching systems using a customizable Markov chain model," *Journal of Network and Computer Applications*, vol. 130, no. November 2018, pp. 39–51, 2019.

[4] G. Bai and C. Williamson, "Workload characterization in Web caching hierarchies," *Proceedings - IEEE Computer Society's Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, MASCOTS*, vol. 2002-Janua, pp. 13–22, 2002.

[5] R. Palmer and M. Utley, "On the modelling and performance measurement of service networks with heterogeneous customers," *Annals of Operations Research*, vol. 293, no. 1, pp. 237–268, 2020.

[6] R. Shi, Y. Gan, and Y. Wang, "Evaluating scalability bottlenecks by workload extrapolation," in *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, (Milwaukee, WI, USA), pp. 333–347, IEEE, 2018.

[7] A. Kattepur and M. Nambiar, "Service demand modeling and performance prediction with single-user tests," *Performance Evaluation*, vol. 110, pp. 1–21, 2017.

[8] S. Bagchi, "The Modeling Approaches of Distributed Computing Systems," in *Communications in Computer and Information Science*, vol. 257, pp. 479–488, 2011.

[9] M. Frincu, B. Irimie, T. Selea, A. Spataru, and A. Vulpe, "Evaluating Distributed Systems and Applications Through Accurate Models and Simulations," *Studies in Big Data*, vol. 36, pp. 1–18, 2018.

[10] Y. Li, Y. Gupta, E. L. Miller, and D. D. Long, "Pilot: A framework that understands how to do performance benchmarks the right way," in *IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 169–178, IEEE, 2016.

[11] L. A. Alpha, "Using remote cache service for Bazel," *Communications of the ACM*, vol. 62, no. 1, pp. 38–42, 2019.

[12] S. Pratt, *Distributed Systems*, vol. 33. 1991.

[13] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*. Amsterdam, The Netherlands: Pearson, fourth ed., 2008.

[14] N. J. Gunther, "How to quantify scalability, the universal scalability law (usl)," Feb 2020.

[15] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. 5th ed., 2012.

[16] P. Rodriguez, C. Spanner, and E. W. Biersack, "Analysis of web caching architectures: Hierarchical and distributed caching," *IEEE/ACM Transactions on Networking*, vol. 9, no. 4, pp. 404–418, 2001.

[17] S. Traverso, P. Torino, M. Ahmed, M. Garetto, P. Giaccone, and S. Niccolini, "Temporal Locality in Today's Content Caching: Why it Matters and How to Model it," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 5, pp. 5–12, 2013.

[18] A. Montazeri, N. R. Beaton, and D. Makaroff, "LRU-2 vs 2-LRU : An Analytical Study," pp. 571–579, 2018.

[19] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques For Experimental Design Measurements Simulation And Modeling*. 1991.

[20] E. D. Lazowska, *Quantitative System Performance Computer System Analysis Using Queueing Network Models*. New Jersey: Prentice-Hall„ 1984.

[21] M. Harchol-Balter and M. Harchol-Balter, *Performance Modeling and Design of Computer Systems-Queueing Theory in Action*. 2013.

[22] Y. C. Tay, *Analytical performance modeling for computer systems*, vol. 7. 2018.

[23] K. Rahimizadeh, M. Analoui, P. Kabiri, and B. Javadi, "Performance modeling and analysis of virtualized multi-tier applications under dynamic workloads," *Journal of Network and Computer Applications*, vol. 56, pp. 166–187, 2015.

[24] A. O. Allen, *Probability, Statistics, and Queueing Theory With Computer Science Applications*. San Diego, CA, USA: Academic Press, second ed., 1990.

[25] N. J. Gunther, "Simple capacity model of massively transaction systems," 1994.

[26] T. Rak, "Response Time Analysis of Distributed Web Systems Using QPNs," *Mathematical Problems in Engineering*, vol. 2015, 2015.

[27] L. Jiang, X. Chang, J. Mišić, V. B. Mišić, and R. Yang, "Performance analysis of heterogeneous cloud-edge services: A modeling approach," *Peer-to-Peer Networking and Applications*, vol. 14, no. 1, pp. 151–163, 2021.

[28] X. Chang, R. Xia, J. K. Muppala, K. S. Trivedi, and J. Liu, "Effective modeling approach for iaas data center performance analysis under heterogeneous workload," *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 991–1003, 2018.

[29] M. Awad and D. A. Menascé, "Deriving parameters for open and closed QN models of operational systems through black box optimization," *ICPE 2017 - Proceedings of the 2017 ACM/SPEC International Conference on Performance Engineering*, pp. 127–138, 2017.

[30] P. J. Kuehn, "Approximate Analysis of General Queuing Networks Decomposition," in *IEEE Transactions on Communications*, vol. COM-27, (Melbourne, Australia), pp. 113–126, 1979.

[31] J. Vilaplana, F. Solsona, I. Teixidó, J. Mateo, F. Abella, and J. Rius, "A queuing theory model for cloud computing," *Journal of Supercomputing*, vol. 69, no. 1, pp. 492–507, 2014.

[32] A. M. Mohamed, L. Lipsky, and R. A. Ammar, "Performance Modeling of a Cluster of Workstations," *Proceedings of the International Conference on Communications in Computing*, no. May 2014, pp. 227–233, 2003.

[33] T. Bérczes and J. Sztrik, "Performance modeling of proxy cache servers," *Journal of Universal Computer Science*, vol. 12, no. 9, pp. 1139–1153, 2006.

[34] I. Bose and H. K. Cheng, "Performance models of a firm's proxy cache server," *Decision Support Systems*, vol. 29, no. 1, pp. 47–57, 2000.

[35] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "Analytic modeling of multitier Internet applications," *ACM Transactions on the Web*, vol. 1, no. 1, 2007.

[36] P. A. 2014-2021, "Overview: What is prometheus?." `https://prometheus.io/docs/`. Accessed: 2021-05-26.

[37] N. J. Gunther and S. Moeding, "Analyze system scalability with the universal scalability law," Feb 2020.

A

Experimental Setup Configuration

Table A.1: Configurations and results of the experimental setup.

Begin of Table					
VM	concurrency	bb-workers	Workload	Time(s)	AvgThroughput(ops)
1x1CPU	4	4	100	229.920	16.000
1x1CPU	4	4	200	277.639	27.000
1x1CPU	4	4	500	439.274	22.000
1x1CPU	4	4	1000	882.031	23.700
1x1CPU	8	8	100	227.865	18.000
1x1CPU	8	8	200	271.752	25.600
1x1CPU	8	8	500	455.779	25.600
1x1CPU	8	8	1000	693.635	23.333
1x1CPU	16	16	100	229.209	9.000
1x1CPU	16	16	200	293.146	10.400
1x1CPU	16	16	500	432.853	18.200
1x1CPU	16	16	1000	849.869	17.800
1x1CPU	32	32	100	231.327	5.400
1x1CPU	32	32	200	309.547	6.400
1x1CPU	32	32	500	549.627	10.300
1x1CPU	32	32	1000	953.946	12.050
1x2CPU	4	4	100	145.291	23.500
1x2CPU	4	4	200	185.941	23.500
1x2CPU	4	4	500	290.947	16.600
1x2CPU	4	4	1000	471.205	36.400
1x2CPU	8	8	100	152.765	23.500
1x2CPU	8	8	200	174.914	37.600
1x2CPU	8	8	500	280.733	39.200
1x2CPU	8	8	1000	478.982	26.200
1x2CPU	16	16	100	145.060	33.400
1x2CPU	16	16	200	178.793	24.500
1x2CPU	16	16	500	277.835	40.600
1x2CPU	16	16	1000	496.280	36.400
1x2CPU	32	32	100	143.960	20.000
1x2CPU	32	32	200	181.905	28.667
1x2CPU	32	32	500	287.265	17.000
1x2CPU	32	32	1000	487.875	16.800

Continuation of Table A.1					
VM	concurrency	bb-workers	Workload	Time(s)	AvgThroughput(ops)
1x4CPU	4	4	100	367.168	2.800
1x4CPU	4	4	200	563.544	12.800
1x4CPU	4	4	500	569.635	9.600
1x4CPU	4	4	1000	940.755	14.700
1x4CPU	8	8	100	273.699	14.250
1x4CPU	8	8	200	275.275	17.600
1x4CPU	8	8	500	403.407	21.700
1x4CPU	8	8	1000	630.227	20.700
1x4CPU	16	16	100	231.042	14.200
1x4CPU	16	16	200	271.429	25.750
1x4CPU	16	16	500	325.336	41.167
1x4CPU	16	16	1000	517.434	56.200
1x4CPU	32	32	100	202.058	28.000
1x4CPU	32	32	200	220.864	30.667
1x4CPU	32	32	500	268.720	76.200
1x4CPU	32	32	1000	379.120	73.600
2x1CPU	2	4	100	232.829	28.400
2x1CPU	2	4	200	239.570	22.000
2x1CPU	2	4	500	307.520	44.400
2x1CPU	2	4	1000	454.184	30.600
2x1CPU	4	8	100	178.878	21.500
2x1CPU	4	8	200	217.833	20.400
2x1CPU	4	8	500	299.895	20.600
2x1CPU	4	8	1000	445.794	19.100
2x1CPU	8	16	100	184.807	21.000
2x1CPU	8	16	200	224.889	21.400
2x1CPU	8	16	500	295.779	46.900
2x1CPU	8	16	1000	468.985	20.000
2x1CPU	16	32	100	176.468	19.500
2x1CPU	16	32	200	222.474	35.000
2x1CPU	16	32	500	299.541	45.200
2x1CPU	16	32	1000	461.415	20.600
2x1CPU	32	64	100	186.682	19.000
2x1CPU	32	64	200	197.277	20.500
2x1CPU	32	64	500	325.479	24.889
2x1CPU	32	64	1000	535.532	27.000
End of Table					

B

The USL package in R

Below is the R code used to model the data using the Universal Scalability Law.

```
// Performance Evaluation using USL
library(usl)
//Define the dataset
concurrency=c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
throughput=c(8.09, 15.05, 21.53, 26.36, 30.41, 33.45, 35.88,
37.23, 39.27, 40.20, 40.61, 41.63)
specsdm=data.frame(concurrency,throughput)

//Use the usl model in the dataset
usl.model<-usl(throughput concurrency,specsdm)

print("Summary")
print(summary(usl.model))

efficiency(usl.model)
barplot(efficiency(usl.model), ylab="efficiency/concurrency",
xlab="concurrency")

//Predict throughput for higher concurrency level
predict(usl.model, data.frame(concurrency = c(16, 32, 64)))
```

Output:

```
[1] "Summary"

Call:  usl(formula=throughput   concurrency, data=specsdm)

Efficiency:
 Min     1Q      Median  3Q      Max
 0.4187  0.5162  0.6457  0.8130  0.9763


Residuals:
 Min       1Q        Median   3Q       Max
 -0.51900  -0.21361  0.05938  0.15826  0.33281
```

```
Coefficients:
        Estimate  Std.  Error  t value  Pr(>|t|)
 alpha  0.0680652   0.0075417    9.025   8.35e-06  ***
 beta   0.0049927   0.0004278   11.672   9.75e-07  ***
 gamma  8.2861415   0.1361476   60.861   4.40e-13  ***
--

Signif.codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.'  0.1 " 1

Residual standard error:  0.2835 on 9 degrees of freedom

Scalability bounds:
 limit:  throughput 121.7 (Amdahl asymptote)
 peak:   throughput 41.54 at concurrency 13.66
 opt:    throughput 41.46 at concurrency 14.69

     1          2          3          4          5
 0.9763290  0.9081428  0.8661048  0.7953038  0.7339966
     6          7          8          9         10
 0.6728101  0.6185888  0.5616305  0.5265820  0.4851474
    11         12
 0.4455413  0.4186709


Predicted Throughput for higher concurrency level:
    1         2         3
 41.18315  32.88633  20.86299
```