THESIS

HORIZONTAL SCALING OF VIDEO CONFERENCING APPLICATIONS IN VIRTUALIZED ENVIRONMENTS

Submitted by

Mante Luo

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2016

Master's Committee:

Advisor: Shrideep Pallickara

Christos Papadopoulos
Daniel Turk

ABSTRACT

HORIZONTAL SCALING OF VIDEO CONFERENCING APPLICATIONS IN VIRTUALIZED ENVIRONMENTS

Video conferencing is one of the most widely used services in the world. However, it usually requires dedicated hardware and expensive licenses. Cloud computing has helped many companies achieve lower operation costs, and many applications including video conferencing are being transitioned into the cloud. However, most video-conferencing applications do not support horizontal scaling as a built-in feature, which is essential to embrace the advantages of virtualized environments. The objective of this thesis is to explore horizontal scaling of video conferencing applications. We explore these ideas in the context of a Jitsi an open-source video-conferencing. The thesis develops a methodology for horizontal scaling in the Amazon EC2 cloud with the objective of ensuring quality of service such as per-packet latency (primarily), loss rates, jitter, and the number of participants per session. We build predictive models to inform our horizontal scaling decisions. Proactive scaling allows us to preserve several qualities of service metrics for video-conferencing. Scaling in the EC2 environment is fast and cost-effective with the added benefit of high availability, which helps us support large number of users consistently without much downtime.

ACKNOWLEDGEMENTS

I want to sincerely thank my advisor Dr. Shrideep Pallickara for his guidance and patience over my master study. And also my late father, his encouragement was my very first motivation to pursue further study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Motivation

Video-conferencing is one of the most widely deployed services in the world. It is used by millions of people every day from all kinds of devices. Enterprises use it to maintain communication between offices from multiple locations. People can call their families even half a world away. At least in the foreseeable future, the demand for videoconferencing will stay and may become even higher.

However, most video-conferencing solutions require dedicated hardware that often comes with quite expensive licenses. Most importantly, this software is hard to upgrade. On the other hand, real-time multimedia transmission is very expensive—especially in many commercial solutions in which sessions must be mixed before they can be sent to the users. Video mixers cannot be easily deployed to commodity servers because they require special multicast mechanisms and mixing processes.

It is desirable to be able to easily deploy video-conferencing servers to machines that we use every day and to easily upgrade systems when necessary. Thankfully, a few promising solutions are now emerging in the open-source world. They are designed with ease of deployment in mind and often exhibit some very modern architectural designs. Nevertheless, these solutions do not normally provide horizontal scaling, but the modularity they have makes it easier to modify and transform.

Cloud-computing platforms such as Amazon Web Service, Rackspace and Microsoft Azure have become more and more popular in recent years, and they have become quite mature with many DevOps tools.[1] Therefore, they are the easiest tools to buy and to start using. Best

of all, they bill by the hour. In addition, their elasticity comes in handy: cloud services make it possible to start up new servers when the need arises and shut down immediately if they are no longer required. [2] The aim of this thesis is to transform a famous open-source video-conferencing project—namely, Jitsi—into a horizontally scalable project that can be deployed in the EC2 environment.

Many previous studies have worked on predicting the overheads for applications in virtualized environments. Often, micro-benchmarks are performed in virtual machines—either on single VM or in the consolidated servers—and different predictive models are used for training, linear regression, ANNs, SVM, and other applications. Many kinds of application are on the list, the most noticeable being web stacks, filestore, and Map-Reduce applications that encompass most aspects of systems including CPU, memory, disk, and network IO. Also, different technologies like XEN, KVM, and ESX are used for this study. Nevertheless, video-conferencing is a field that is not widely studied in the literature concerning virtualization impacts, though study of it is badly needed due to its popularity. Our work focuses on modeling the impacts of system resources and video-conferencing metrics, which will help us to determine the appropriate time to scale horizontally in the Amazon Elastic Compute Cloud.

With the help of open-source projects like Jitis and the powerful Amazon Cloud, we are able to deploy video-conferencing applications in commodity servers and to conduct horizontal scaling. However, there are still some questions to be answered. This thesis investigates two research questions: 1) When is it good to scale horizontally to maintain a quality video-conferencing service? 2) How can we scale in the Amazon Cloud to ensure that the system is running with high availability?

1.2 Brief Description of the Research Design

This thesis investigates a popular open-source project for video-conferencing called Jitsi, which is currently maintained by the software giant, Atlassian. This application is a good example of engineering in the open-source world, especially because of its modern structure, which provides both video and audio conferencing through a video mixer or bridge. We carefully analyzed the stack and picked up the parts that are easily scalable. We ran it under Amazon Elastic Compute Cloud, which is the industry leader and is adopted by most companies. Loads were generated to make pressure on the server. Meanwhile, we gathered all the metrics—both system level to application specific—and stored them for analysis. Models were built by applying multi-variate linear regression, thus making it possible to predict their impacts on the video-conferencing system before users successfully join the conference, which helps to scale the mixer at the right time before limitations happen. After setting certain thresholds to vital metrics like bandwidth, jitters, memory-usage rate and so on, we can auto-scale to new servers when needed and can eventually migrate conferences between nodes when necessary.

This thesis focuses on video-conferencing applications, which are well-known for their complexity due to their mix of CPU, memory and network resources. Multimedia, real-time transmissions—especially WebRTC and RTP protocols—can generate rather different outcomes according to the underlying resource allocated. Applications can behave differently when deployed in a virtualized environment, which can lead to extra costs if it becomes desirable to transit to cloud or virtualization technologies. For this reason, we built a set of models to try to see through the relationships of impacts between different metrics.

As a result, we are able to transfer a video-conferencing application that used to be hosted on dedicated servers to commodity servers that can spawn up new nodes when necessary and so ensure quality user experience.

1.3 Thesis Outline

Chapter 2 explains the background of different video-conferencing architectures and our chosen video-conferencing application, Jitsi. It also introduces overheads due to virtualization. Multimedia transmission protocols such as WebRTC and RTP are investigated—especially the control protocols that provide us with statistics. Also, related work is explored at the end. Chapter 3 describes the research design we used to monitor, model, and scale the video-conference system. This research design helps us answer the research questions that are proposed at the beginning of the chapter. Chapter 4 reveals the results derived from the application of the techniques described in Chapter 3. Comparisons are made and analyses are performed to prove the improvements of introducing the models. Conclusions and future work are discussed in Chapter 5, which confirms the improvements because of our system design.

2. BACKGROUND

Before we can consider the details of the scaling system, it is necessary to offer some background about video-conferencing architectures by comparing traditional and modern designs. WebRTC and RTP are protocols that transmit video sessions that introduce complexity to the video-conferencing system. On the other hand, overheads are inevitable in virtualized environments. Predictive models are trained to help us scale ahead of time. A literature review is offered at the end of the chapter.

2.1 Traditional video-conferencing architecture

Video-conferencing applications have been deployed over the last a few decades, so they are very mature, in a way. Video-conferencing solutions tend to be quite costly, but they are essential to enterprise. Naturally, companies try to reduce expenditures even as they attempt to maintain the same service quality. However, this goal is hard to achieve, and often new architectures are needed to modernize the service.

For a long time, video-conferencing solutions required expensive, dedicated servers to operate as part of an enterprise solution. Companies have to purchase hardware and host services on it without much flexibility and control over the underlying resources and scalability. Essentially, they have to prepare for the highest loads and pay extra costs for unneeded servers. Operation expenditure can be more costly if it becomes desirable to upgrade the system, which leads to buying more physical machines.

It is essential that everyone in a video conference can see every other participant's face and hear their voices. In one solution, video mixers sit in the center, accepting all sessions, combining the videos and sending them to everyone. This whole process consumes a huge

amount of computational resources, which explains why the providers often require special servers for deployment.

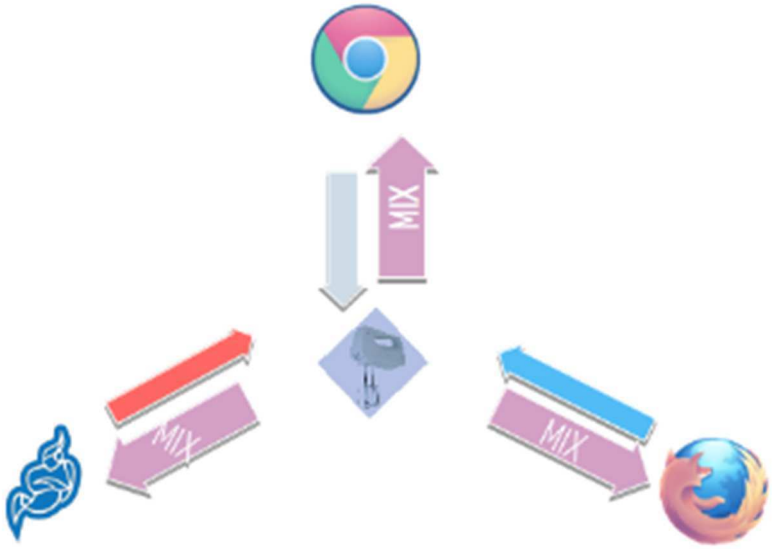


Figure 1- Traditional Video Mixer Architecture

Mixing can save bandwidth and CPU resources on the customer's side. But because the computing power increases so drastically year by year, even a mobile phone is capable of what a desktop used to do. This kind of architecture hinders the path to using commodity servers and to ease of upgrading when the need comes.

2.2 Modern Structure of Video-conferencing applications

Apart from the traditional architecture, there are many emerging solutions that try to solve the problem from another direction. Video-conferencing applications have gradually come to embrace the relaying method—especially with the growing trend of cloud-computing and virtualization technologies. Many companies buy services from cloud providers or host a scalable solution on their private cloud. All of these services combined help companies to reduce operation costs and to achieve better control when the need changes, because they make it

possible to adjust costs to fit the amount of services that are needed and possibly to shut down a few applications when they are no long needed.

This thesis uses an open-source solution called Jitsi, which was started at the University of Strasbourg, France. Jitsi is currently maintained by Atlassian, the company behind Bitbucket. [3] Jitsi is a modern audio/video-conferencing solution that adopts WebRTC and XMPP underneath to achieve the best modularization and easiest upgrade in the open-source world. There are also many excellent solutions like Join-me in the industry, but their codes are closed such that they do not suit our research needs.

The structure of Jitsi consists of a web server, an XMPP server, a focus component, and possibly multiple video bridges. First, the user interface works through a browser that employs WebRTC, which means that everyone with a browser can use it without restrictions. The service is provided through an Nginx web server. Then it is the XMPP (Extensible Messaging and Presence Protocol) server. Jitsi uses XMPP to connect each component and to provide basic text-chatting. The Focus component serves an essential role: to guide audio and video streams to the video bridge for latter relaying process. Last but not least, the video bridge handles stream mixing and relaying: the real core functionality of the Jitsi solution.

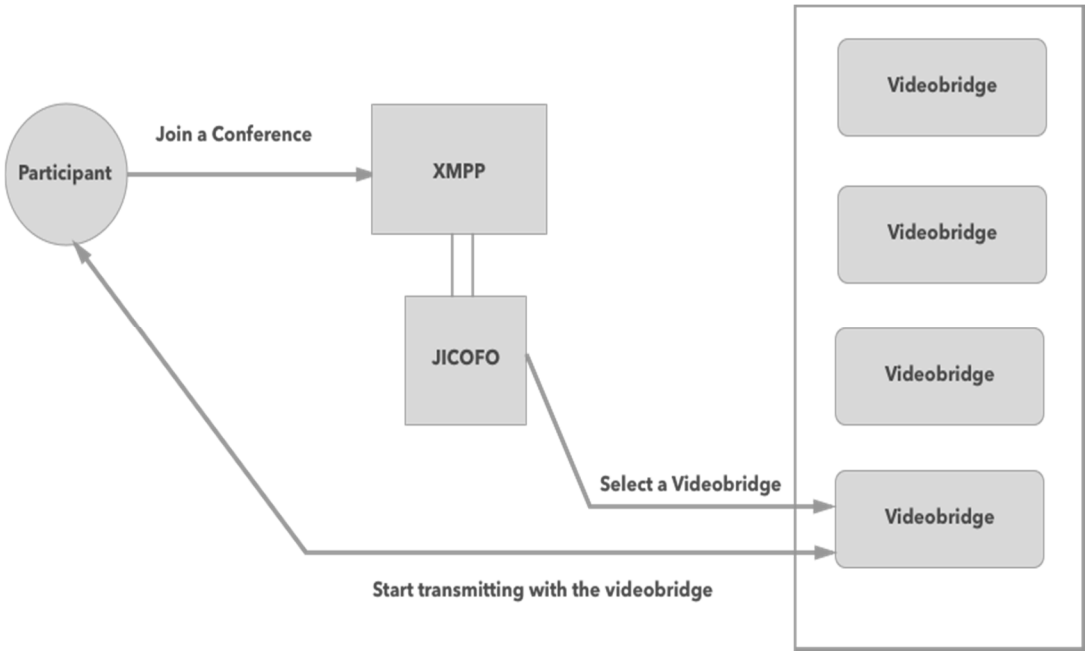


Figure 2- Jitsi System Basic Flow

The flow begins as a user joins a conference with a WebRTC protocol on the web user interface. WebRTC protocol is used to establish the request to join the MUC (multi-user Conference), which is then converted to XMPP's format as a video or audio extension by Jingle extension. Jingle was designed by Google to address the lack of real-time transmission, such as Voice Over IP or video-conferencing communications. Jingle can be considered the XMPP way to transmit multimedia data. RTP is the actual transport protocol underneath.[4]

After the user joins the conference, a special user called Focus appears, whose main responsibility is to allocate transmission channels onto video bridges and back to the user. The Focus component is just another XMPP component that appears like a normal user in a MUC. The only difference is that COLIBRI (Conferences with Lightweight Bridging), an extension of XMPP, makes it possible to add remote media mixers to the XMPP framework. XMPP is very extensible and has been widely used in many different applications[5][6][7], only one of which is

video-conferencing. As a result, it is now possible to start new video bridges on other servers or virtual machines and to become available to incoming users.[3]

Figure 3- Modern Video-conferencing Design

Video bridges or mixers are the most import component of most mainstream video-conferencing solutions, including Jitsi. They constitute that through which multimedia streams with different bandwidth capacities pass, either mixing with or replying to every participant in the conference. The Jitsi Videobridge mixes all the audio streams and replays the video streams. This decision makes it possible for Jitsi to run on commodity machines with enough network bandwidth. Audio mixing is quite mature today, while video encoding and decoding implementation are inefficient. As a result, CPU resources often burn down. And this is not unusual. Commercial solutions like Join-me involve the same trade-off.

After multimedia channels are allocated on the bridge, users can send and receive audio and video streams from the bridge. On the other side, bridges can adjust the output streams with

different transmission rates or qualities, according to the user's current condition. This is what makes it possible for all participants in a conference to enjoy a smooth and quality experience.

2.5 WebRTC and RTP protocol

Despite the fact that Jitsi uses XMPP as its central control component, audio/video streams actually travel in WebRTC and RTP sessions. For this reason, it is necessary to study the protocols and their characteristics.

WebRTC has become quite popular in recent years. The project was started by Google in May of 2011 as a solution for browser-based, real-time communication, which was later standardized in IETF. WebRTC allows browsers not just to request from servers but to initiate real-time transmission between them. Its popularity makes it a perfect choice for video/audio sessions. It is also platform independent[8], and many video-conferencing implementations are based on WebRTC besides Jitsi, such as [9].

On the other hand, Real-time Transport Protocol, standardized in RFC 3550, has existed for decades. It first appeared to meet the needs of multi-media transmission, because primitive transport protocols such as TCP and UDP could not address the problem of out-of-order packet arrival and the tight, real-time requirement for transmission. While RTP started as a dedicated, application-level protocol built on top of UDP, it soon became the choice for audio and video transmissions. [10]

UDP was chosen as the transport protocol instead of TCP because UDP provides only the concept of port. On the other hand, TCP has its own delivery policy and quality-of-service control, which can bring too many overheads for multi-media transmission. RTP, alongside RTCP (RTP Control Protocol), solves the problem quite nicely. [4]

RTCP always comes with RTP, with the two ports one after another.[10] Because it is built on UDP, all the quality control is the burden of the protocol implementer, which is why RTCP is very important: it provides the vital metrics of how well a multi-media session is running. Essentially, the RTCP packet is sent periodically, carrying information—about packet counts, packet loss, packet-delay variation, and round-trip delay time—that is used by applications to determine quality-of-service parameters, to perform maintenance, or perhaps to change to another codec.

In a bigger picture, all multi-media streams are established as RTP sessions, carried through RTP, and controlled using the information provided by RTCP. This is exactly how video-conferencing applications work, and Jitsi is no exception. Jitsi Videobridge is able to measure the bandwidth and latency of participants. In order to make sure that everyone enjoys quality service, it changes codecs and relaying policy accordingly. If the user has a rather slow Internet connection—because he or she is using a mobile device, for example—lower-quality streams are transmitted and vice versa. Jitsi Video Bridge implemented a Last-N mechanism to reduce transmission overheads while trying not to impact conferencing experience. However, nothing can be achieved if not with the help of RTCP.

Overall, users join conferences through WebRTC protocol, which is transformed to RTP afterwards. For this study, we ran Jitsi under virtualized servers—EC2 instances, specifically—generating enough loads by constantly establishing and removing conferences of different sizes. In the meantime, both system-level and RTCP metrics were gathered and analyzed to build a predictive system for the sake of scalability. The next section explains how we built the models so that the right timing could be found for when to scale to new nodes, thereby to minimize performance degradation when the system limits are reached.

2.5 Horizontal Scaling

When coping with increase of loads, instead of restarting down servers after adding more computing resources, the better way is to start up new servers of the same pricing level. This is also called scaling out, which contrasts with scaling up that upgrades on the same server.

A strong motivation of horizontal scaling is the reduction of operation costs and high availability. In the virtual server market, it usual costs twice the price whenever the need of adding more memory or CPU comes. However, the upgrading of server tier does not guarantee an equivalent increase in performance. Most noticeable differences are that network IO and CPU may not be doubled the same way as memory does. Hence, under the same budget, starting up more servers can provide more network IO and computing power, compared with a single high performance machine.

Cloud provider like Amazon has mastered and standardized many virtualization technologies, and makes horizontal scaling an efficient choice for many enterprises. To deal with spikes caused by workload patterns, Cloud customers can scale out accordingly at the appropriate time, to maintain quality user experience, and shut down servers when necessary. Most virtual servers are charged by hour and on-premise infrastructure is no long needed.

Complexity applications like video-conferencing can leverage horizontal scaling to react according to needs and exploit computing resources that come with multiple servers. And most importantly, we can reduce operation costs considerably by utilizing the elasticity.

2.4 Overheads due to Virtualization

Virtualization is a technology that creates an abstract over shared hardware resources such as CPU, memory and IO, which to the virtual machines' points, they think they have the complete control. This abstraction creates isolation between virtual machines.

However, virtualization does not come without overheads [11][12]. Essentially, resources are still shared in some way, which makes contention unavoidable. Hence, no matter how sophisticated the virtualization implementation is, overheads can still be present, big or small. Because we picked the Kernel Virtual Machine for our study, it is necessary that we study it to determine its limitations and threshold for scalability.

Many works have been written about overheads in the literature. Important works include [13][14][15]. Applications are run in physical and virtual machines to compare the differences and characteristics caused by the environments. The process often begins with a running CPU and memory and IO-bound benchmarks, and it gathers metrics from different environments. A model is usually built into a utility for decision makers before they transit their applications to the virtualized environment. It becomes clear that applications with different concentrations—one may be computationally intensive, another may be memory-bound, still another might exhaust IO quite intensively—tend to appear with different overheads. As for mixtures, the outcome is often more complicated [14].

Despite the fact that most VMM implementations try hard to minimize the differences—especially for disk and network IO—there are still some overheads. In essence, CPU and memory are two resources for which sharing mechanisms are provided at the hardware-level. So it is possible to assign two cores to this VM and one core to another, and probably each for 2GBs of RAM. But every time simple applications combine with IO, things become complicated. Disk and network can be hard to share, and different vendors guarantee performance only for those who use their own drivers—which is a big problem when it comes to virtualization. Smart implementers came up with the idea of para-virtualization, which, instead of virtualizing the IO devices, reuses the driver. XEN is famous for its para-virtualization technologies, which provide

a way to improve efficiency over the full-virtualization method—especially in disk and network IO. The outcomes are very noticeable; however, there is still a difference with the bare-metal environment. This is all due to the nature of virtualization: i.e., hardware, network and disk devices are shared among multiple consumers, which makes contention inevitable.

Video-conferencing applications are often quite complex due to the nature of mixed usages of different resources. First, encoding and decoding are inevitably used to save bandwidth, which is a huge gain on CPU and memory. Besides, as a video bridge to relay the packets, the server consumes a large amount of network bandwidth. And when memory reaches its limit, disk swaps always come in. As a result, the combination can affect the user experience when multi-media transmission is happening. The next section will consider the transport protocols underneath and figure out the metrics-reporting protocols to be used for our study. In this way, we can know the service quality on the user-end when the limits of virtual machines are reached.

2.6 Predictive Models

Because video-conferencing applications are becoming so complicated, we want a way to predict possible outcomes before multi-media streams get allocated, as a means to avoid dragging the whole system down. Much has been done to study such models, and the results are quite inspiring.[16] This thesis starts with linear-regression models and investigates ANNs at the end.

Linear regressions are usually considered the basic methods before something more advanced is attempted, and they work well most of the time. Much of the literature has adopted linear regression, and the results are quite acceptable. Besides, if the relationships are proven to be suitable for linear models, the computation tends to be much less intense than in nonlinear techniques such as artificial neural networks.

However, there are a few problems regarding modeling when many predictors are involved, mostly because not every predictor is related to our response. As a result, prediction accuracy may not be very high, even though more predictors always come with higher R-squared values. With that said, predictor selection or shrinkage is necessary, which not only makes us concentrate on related predictors, but also enables us to compute a lot faster.

LASSO is a shrinkage method that is widely applied to linear regression modeling. It is very powerful, as it cancels unrelated predictors by making their parameters to zero. [17] There are 35 metrics in our research that compass most aspects of operating systems and video-conferencing applications. Not everyone can contribute to our desired model. After LASSO, our modeling tends to be more concise.

Essentially, linear regression is a way to statistically train a model with a trend of linear relationships among predictors, which is quite common in many fields.

However, there are many nonlinear methods when it comes to model training. Prominent among them are artificial neural networks, which are very good at image and voice recognition. We applied ANNs to our gathered metrics. But, it fails to build a significantly better model than the linear methods. Besides, like other advanced methods, it is often too computationally intensive.

2.6 Related Work

Modeling and analyzing applications in virtualized environments are popular topics in the literature. Ever since technologies like XEN and ESX emerged, researchers began to notice different levels of overheads in various computing resources due to the hardware abstractions virtualization made. [18] Studies of possible migration costs became quite necessary for those who were to embrace the new infrastructure, as well as for applications in the virtualized

environments. In the past decade, the focus was on multiple virtual machine hypervisors [19], and especially on applications of different resource impacts running on them. [20][21][15] have conducted experiments in the ESX environment: an industry-strength hypervisor from VMWare. On the other hand, [22][23] [24][25] are focus on the XEN hypervisor. Besides, KVM is also very popular, due to strong support from RedHat, which is used in [26][27]. Apart from hypervisors, studies have been done in cloud platforms, such as Amazon EC2[28], which is mentioned in [29][21][31].

Different aspects of the system have been investigated. Disk IO is a very famous topic, which is mentioned in [21][20]. Studies like [22] investigate Network IO overheads in the virtualized environment. Besides the focus on each aspect, different stacks of applications are also tested, such as RUBiS, FileBench, MapReduce, and the like.

With respect to scaling, research has focused on different applications in the Cloud. [32] describes the general cloud-computing model for horizontal scaling, while [33] is concerned about scaling with Map-Reduce. [34] uses probabilistic models to determine scaling for a NOSQL cluster.

XEN is the most famous choice for researchers in the literature, not only because it is the first open-source solution on the market, but also because it is a slick model that makes isolation easy for researchers. XEN is a Type-1, bare-metal hypervisor that generalizes the common APIs all operating systems are supposed to have, and it runs directly on the hardware. Designed using a microkernel with a rather small codebase and serious battle tests and audits, XEN is advertised to provide strong isolation between VMs and good performance, especially with IOs, due to para-virtualization. Our chosen Cloud platform, Amazon EC2, is also XEN based, which is quite stable and cost-effective.

Uyar et al [35] have explored the use of scalable conferencing in distributed brokering environment comprising physical machines; this involved the use of a high-performance messaging substrate [36][37] that supported publish/subscribe and peer-to-peer communications intended for deployment in grid computing environments [38]. On the other hand, but there have not been many studies about video-conferencing systems in the Cloud setting , despite the popularity of video-conferencing and expensive solution costs, which are our focus in this thesis.

3. RESEARCH DESIGN

This chapter explains in detail the methods we used when conducting the experiments. Through the experiments, we were able to answer the research questions concerning when it is most appropriate to scale to maintain quality service and high availability, an how to scale in virtualized environments—in our case, in the Amazon Elastic Compute Cloud. Sections below start by considering the load generator that enables us conduct benchmarks over the Jitsi solution, which later saves to a time-series database. Section 3.2 describes the approach to training models. Section 3.3 is focused on the complete design of the scaling system.

3.1 Load Generator and Metrics Gathering

To determine when to efficiently scale, we need to study the behavior of video-conferencing applications, we must apply enough pressure, or RTP loads, so that we gather enough valid metrics. It is not possible to have real people test the conference joining to reach considerably high loads.

Being a WebRTC video-conferencing application, Jitsi usually provides services through a web-browser interface in which WebRTC requests are transferred into RTP sessions and further processed at the video bridge. We modified a load generator provided by Jitsi called Hammer, which was designed to present a single Jitsi conference room.

To study the characteristics of video conferencing, we need to simulate the appearance of real users who join different sizes of conference rooms as opposed to one big conference. We rewrote Hammer by introducing concurrent features: e.g., spawning a huge number of conferences, and a timeline mechanism that generates loads of a specific size at a certain time. As a result, we were able to toast Jitsi with different profiles of loads, some very light, and some

heavy ones that reached the server's limits. The mixture of profiles guarantees diversity in the modeling process, which further helps us scale horizontally at the right time.

Because Hammer simulates a huge number of participants who are sending and receiving media streams, deployment can be a little different than it is for the other components. According to our experiments, it takes 1.5G of RAM to generate 0.5G of RAM on the video bridge. If deployed in normal EC2 instances, it is not very cost-effective. Fortunately, Amazon provides special nodes called Spot Instances, which are basically unused EC2 nodes. They usually save more than 70% on average, especially for large instances. [39]

After Jitsi stack starts up, it is possible to gradually run Hammer with different timelines or profiles. At the same time, video bridges keep sending metrics to a time-series database called InfluxDB. InfluxDB is written in Golang. Golang is blazingly fast, which reduces the metrics that gather overheads to a minimum. Visualization can happen as we run benchmarks through a popular solution called Grafana. Finally, to further process the metric data, it can be downloaded through REST APIs from InfluxDB, which makes it language-independent. After running different profiles for three hours, we can gather around 10,000 data points for each size of EC2: micro, small, medium, and large. It worked quite well in our experiment and generated quality data for future modeling.

Profiles are the vital component of Hammer, which not only affect the accuracy of the model but also how much further we can squeeze the limits out of the video-conferencing application. There are three parameters to set: when to start, how many participants, and how long it takes for each conference. For simulation, we use random generators that are based on normal distribution as a means to mimic the real workload and touch corner cases. As a result, if

conferences spawn more frequently and with more participants, it is more intense. We prepared six profiles, each of which lasted around 30 minutes.

During the experiments, we found that resources remain rather steady once media channels are allocated but that joining and leaving can cause quite some turbulence, which is exactly the focus of this thesis. Profiles are parameterized to the goal that, without tearing down the system, we keep most participants interleaving and maintain a rather consistent network flow to the video bridges. Below is the setting we found most suitable. However, since we run on different sizes of EC2 instances, the number of conferences can be varied: bigger EC2 instances with heavier loads if simply put.

Table 1- Jitsi Hammer Workload Profiles

Profile	Size mean	Size SD	Duration mean	Duration SD
profile-1	7	2	65	10
profile-2	6	2	50	10
profile-3	5	2	50	10

Each profile is generated multiple times. Each takes about thirty minutes, and each is run ten times. We did not choose the more lightweight ones, because they tend to be irrelevant when it comes to scaling, which happens before thresholds arrive. The parameters were chosen after a long period of experimentation under the rule that they can generate enough consistent loads without killing the video bridges.

During the experiments, we observed the different focus of different profiles. The profiles were chosen after many experiments, because, if conferences are joined too in too scattered a fashion, the metrics can generate too many delays to show the results. If they are too close, the

video bridges are fully occupied too quickly. The above parameters are good at generating loads while not causing too many impacts. Besides, data are generated using normal distribution, so they are actually very diverse.

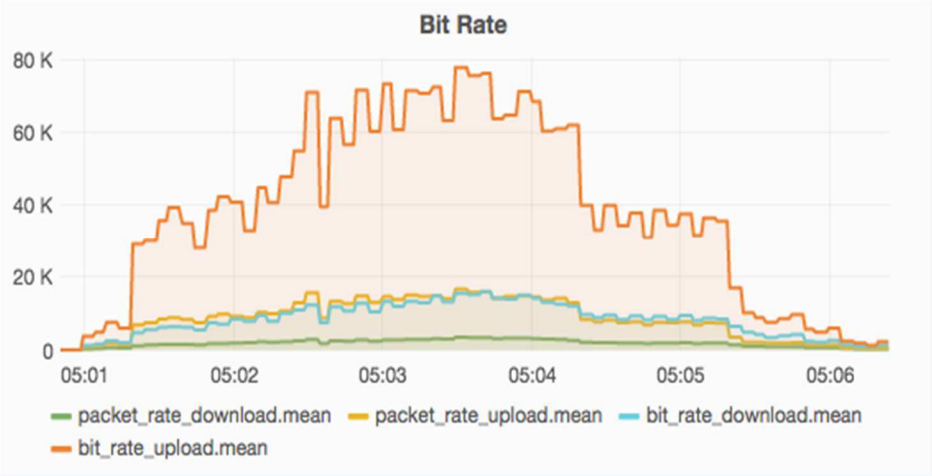


Figure 4- Example of Good Workload

The structure of our load generator, Hammer, is presented below. Hammer is improved from Jitsi Hammer. It simulates conferences with participants who exchange video sessions. It performs well and is written in Java. This works much better than solutions like web-browser automation tools, such as Selenium, that connect to video bridges and behave like real browser users. Selenium provides more real-life simulation, but it requires too many resources when the number of users increases.

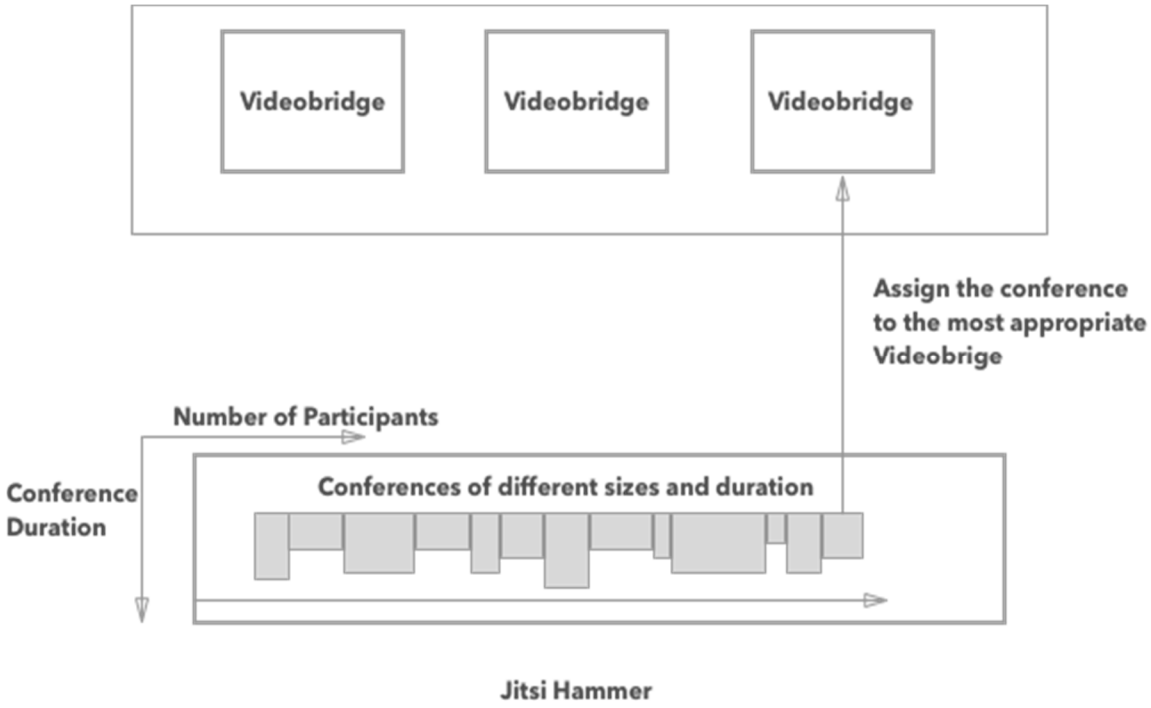


Figure 5- Jitsi Hammer

Hammers initialize each conference at a random time and in a random size. After that, multi-media sessions begin transmission between them and video bridges, which simulate real users in conferences. Each rectangle in the graph above is a conference. The x-axis represents the number of participants, and the y-axis represents the duration of the conference. As time goes by, different conferences are assigned to corresponding video bridges given the generated parameters.

When the load generator hits enough loads, video bridges start to show results. This is where metric gathering happen. In our case, metrics are stored in a popular time-series database called InfluxDB. Furthermore, metrics are sent periodically through REST APIs from each video bridge instance.

One good thing about InfluxDB is that it is designed to handle metric data. It is therefore not necessary to design the schemas to store them as it is in the usual relational databases. Besides, it offers very good support of sampling, which helps to improve storage efficiency.

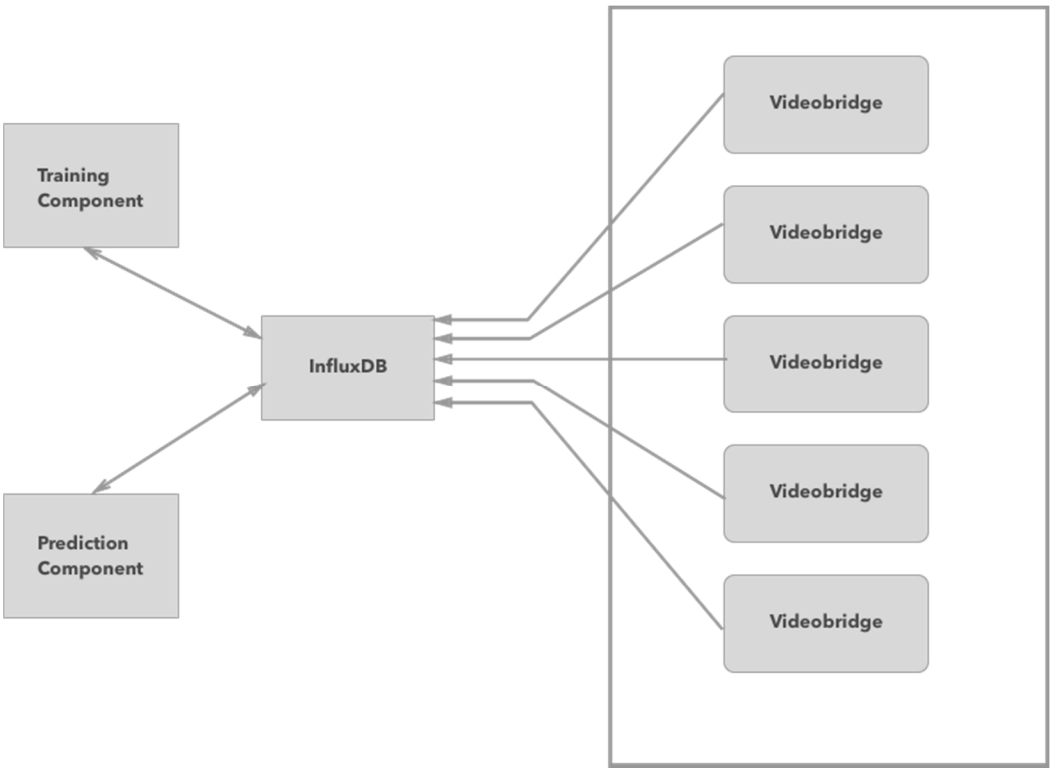


Figure 6- Metrics Gathering and Retrieving

As shown above, we have real-time data stored in the database as the metrics are sending, which makes it possible to visualize the running conditions of our servers.

3.2 Model Training

After the necessary amount of data has been gathered, we can start to build models over different vital metrics by retrieving, from InfluxDB, a powerful time-series database [40]. InfluxDB provides an SQL-like query language and easy-to-use REST APIs, which allow us to use Python as the script language. Compared with Java, Python is more suitable for model training. We used toolkits like Numpy and Scikit-learn and the interactive way to experiment and improve.

First, we tried linear regression models using the python toolkits by trying different shrinkage methods and iterations. We try to predict the resource impacts when new conferences are created so that we can react before system limits are reached, which can cause cascading failures for most participants.

Our concentration is on five vital metrics of system and video-conferencing application. As for predictors, there are 34 in total listed below.

Table 2- Metrics to Gather

Metric	Description
audiochannels	Number of audio channels
avgload_1	1 min average load
avgload_5	5 mins average load
avgload_15	15 mins average load
video streams	Number of video streams
video channels	Number of video channels
used_memory	Used memory in MB
total_udp_connections	Total UDP connections
total_partially_failed_conferences	Total partial failed conferences
total_no_transport_channels	Total number of channels with no transport
total_no_payload_channels	Total number of channels with no payload
total_memory	Total memory in MB
total_failed_conferences	Total number of failed

	conferences
total_conferences_created	Total number of created conferences
total_conferences_completed	Total number of completed conferences
total_conference_seconds	Total seconds of all conferences
total_channels	Total channels
threads	Total number of threads
rtt_aggregate	The aggregate time of Round Return Time
participants	Total number of participants
packet_rate_upload	Upload speed in pkt/sec
packet_rate_download	Download speed in pkt/sec
loss_rate_upload	Upload RTP loss rate
loss_rate_download	Download RTP loss rate
largest_conference	The largest conference in the bridge
jitter_aggregate	Aggregate of jitters
cpu_user	CPU user
cpu_usage	CPU usage
cpu_system	CPU system
cpu_iowait	CPU iowait
cpu_idle	CPU idle

conferences	Total number of conferences
bit_rate_upload	Upload speed in bit/sec
bit_rate_download	Download speed in bit/sec

The above metrics encompass most aspects of a video bridge. They exclude only disk metrics, because conferencing services do not retain data for performance and security concerns. To apply linear models, we set response and predictors as below.

The vital metrics that we use as responses in linear regression are the following: jitter_aggregate, used_memory, and rtt_aggregate. They are the values that appear after the new conference is joined, while all predictors are values before the joining. In this way, we are able to predictor future metrics from the size of the new conference and all metrics at the moment. There are some other metrics that could be of interest, such as CPU usage and Disk IO; however, due to the efficient architecture of Jitsi, they rarely exceed certain limits.

Before training, we need to preprocess the data we gathered and reflect on the change before and after joining new conferences. First, we average all metrics with the same number of participants because impacts do not show up instantaneously but gradually. Secondly, we find the two averaged sets of metrics before and after. Third, we calculate the difference between participants in the video bridge, and we start training.

After preprocessing, each set can be independent. As long as we run enough benchmarks, we are able to train accurate models for prediction. As mentioned in last section, we run three sets of profiles, each generating about ten timelines for Jitsi Hammer. Although each one may not finish at exactly the same time, we should be able to get around 11,000 data points, which can later be pre-processed to around 3000 training points. Normalization is applied before training.

After training and cross-validation, the results are quite promising. Most of the vital metrics we chose to reflect scaling are linear enough with acceptable R-squared values under ten-fold CV. Each table below is for one type of EC2 instance, and each row is trained for one response.

Table 3- T2.Micro Training Results

metrics	R-squared Mean	R-squared stdev.
used_memory	0.9328	0.0072
rtt_aggregate	0.7069	0.0361
jitter_aggregate	0.4880	0.0318

Table 4- T2.Small Training Results

metrics	R-squared Mean	R-squared stdev.
used_memory	0.9361	0.0062
rtt_aggregate	0.6763	0.0461
jitter_aggregate	0.5202	0.0317

Table 5- T2.Medium Training Results

Metrics	R-squared Mean	R-squared stdev.
used_memory	0.9469	0.0195
rtt_aggregate	0.7928	0.0743
jitter_aggregate	0.6668	0.0748

The results are rather promising. Most of the vital metrics we are concerned with exhibit acceptable R-squared values after ten-fold cross validation. We did not consider download RTP loss rate, because it seems insignificant for a video bridge, the main job of which is to route multi-media data. Also, the accuracy is too low to be considered in our linear-regression models.

In the next part, these models are used to predict impacts when new conferences are joining, which helps a lot when it comes to scaling horizontally.

We also tried artificial neural networks for the modeling. Compared with linear regression, artificial neural networks tend to need much more time for training. Most importantly, the results are not significantly better. Below is a table. We ran the data we gathered from t2.small node with ten-fold cross-validation. We ran using 20 sets of data—about 12000 data points—from ten different timelines using Jitsi Hammer.

Table 6- Artificial Neural Network Training Results

Metric Name	R-squared Mean	R Squared stdev.
used_memory	0.9342	0.0224
rtt_aggregate	0.7034	0.0567
jitter_aggregate	0.4812	0.2573

The results of the multi-variable linear regression and ANN regression seem to be very close—at least from the ten-fold cross-validation. However, due to the randomness of ANNs, the standard deviations tend to be larger—especially for complicated responses like jitter, though it is quite stable for linear regression. Besides, training time is also very different. The ANN model tends to run a few times of longer when training the same datasets and results in similar models. Therefore, we choose to use linear regression in our scaling mechanism.

3.3 Scaling System Design

With all the mechanisms available to retrieve data and get predictive models, we are now able to answer the research questions about when to scale efficiently. Horizontal scaling works before currently operational servers are overfeeding, which usually leads to reduction of service quality, as shown in the graph below. By applying prediction models, we are able to scale before

we hit the limits, and the timing helps to maintain the quality of the whole conferencing system when multiple servers are running together.

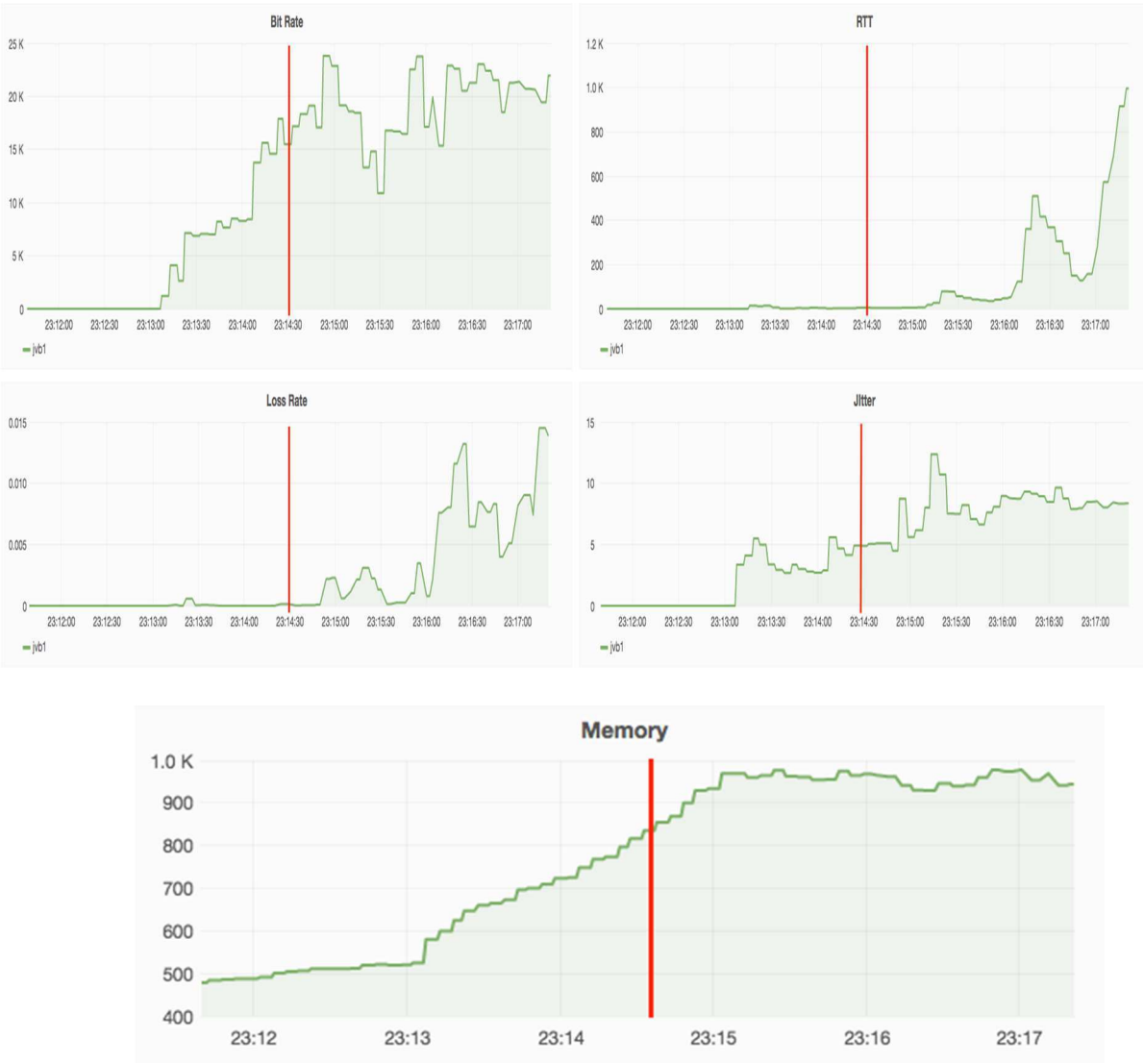


Figure 7- Metrics Before and After Thresholds

However, all of these have to come with a scaling mechanism so that we can start up new servers before one video bridge stops accepting new conferences. In the thesis, we first preset thresholds and then use AWS SDK to spawn new EC2 instances as needed.

Meanwhile, oscillation is inevitable due to the complicated conditions introduced by cloud environments. To cope with it, we tend to scale from 5% to 15% before the thresholds so

that sudden spikes will not impact our system. To scale in the EC2 environment efficiently, we came up with the following design, demonstrating different conditions when we need to assign conferences.

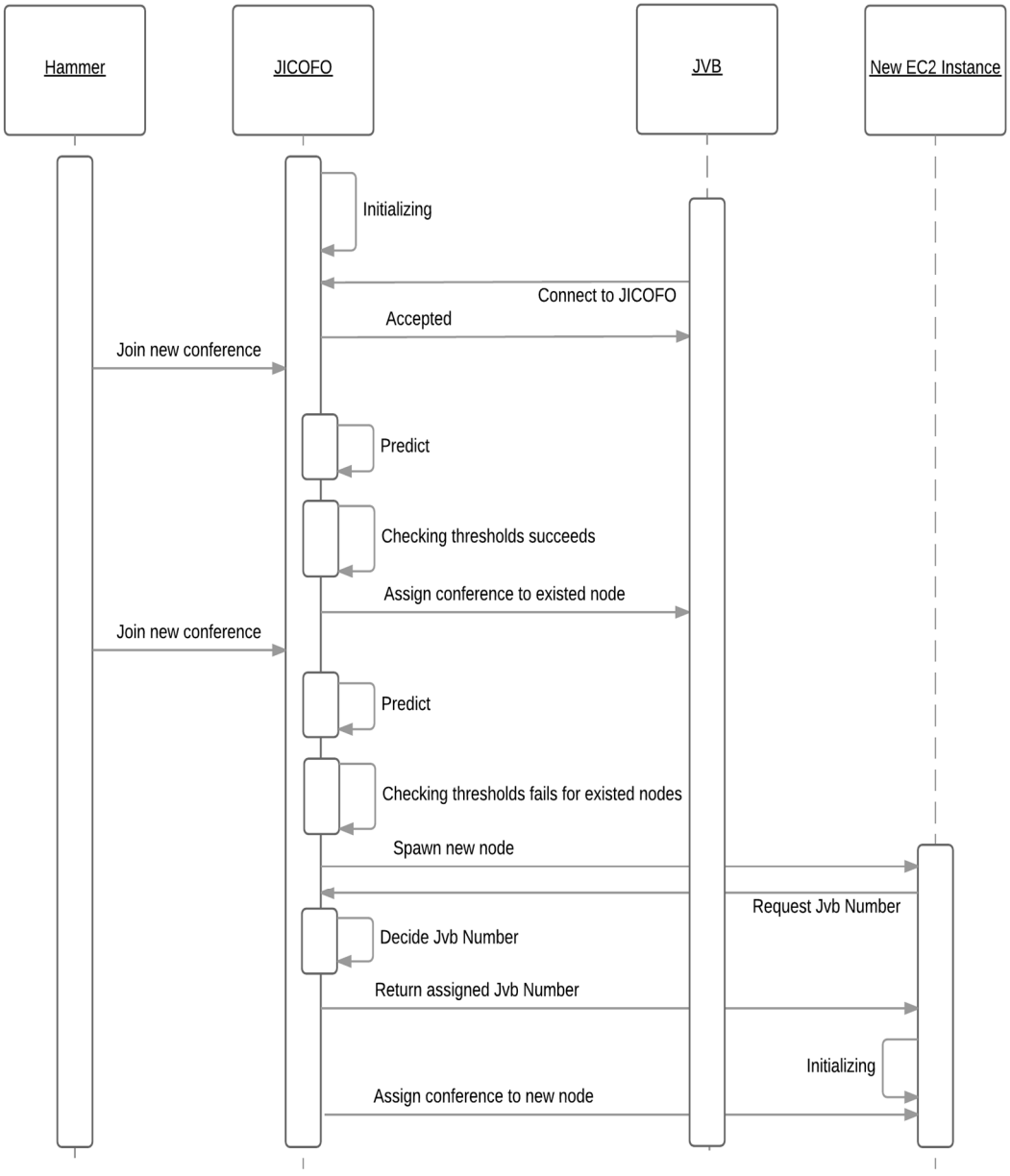


Figure 8- Scaling System Design

Hammer is the load generator. JICOFO is the balancer. JVB is the video bridge. The last one is the new video bridge spawned in EC2. First, JICOFO starts up as the central controller and balancer. Then the first video bridge connects to it and gets accepted to the bridge list. From now on, Hammer generates new conference and consults the JICOFO. The controller conducts prediction and thresholds checking, which turns out to be good enough to assign to the first connected node. At some point later, another conference joins the JICOFO, and the existing video bridges exceed our preset thresholds. As a result, a new bridge is started. It tries to get the JVB number. JICOFO does the calculation and returns a number back to the bridge. The new node then connects to JICOFO and gets assigned to the new conference. This is an overview of what happens when bootstrapping and scaling in the system.

By applying the prediction models we built, we produce another layer of guarantee that a more accurate overview can be used to contribute to the scaling. This way, it behaves very consistently when we try to maintain quality service. The next chapter considers running the system in the Cloud environment and having participants join at different times to achieve horizontal scaling. We will also show the improvements when we know scaling ahead of time.

4. EVALUATION

There are a few aspects of scaling. First, we need to be able to start new servers in the EC2 environment when the running nodes are not enough. This tells us how to scale in the Cloud environment. Second, we compare vital metrics with and without the prediction models and conduct t-tests to show the improvements due to our prediction models. It proves that our design—which involves scaling ahead by applying predictive models—can help reduce contentions when system limits are reached. It also tells us when to scale to provide quality video-conferencing service.

4.1 Experiment Setup

To evaluate our system design, we run the whole stack on the EC2 environment with three different instance types: t2.micro, t2.small and t2.medium. Memory ranges from 1G to 2G to 4G. T2 tier is the baseline for all EC2 virtual machines, which are the suitable choice for deploying video-conferencing applications in affordable commodity servers.

To evaluate, we are running the system based on Jitsi in an EC2 environment. For starters, there is only the XMPP server, the Focus server, and InfluxDB running on one medium node with 4G RAM. This one does not directly handle multi-media sessions but rather handles a balancer and a central controller. Therefore, a t2.medium node is more than enough to provide the service.

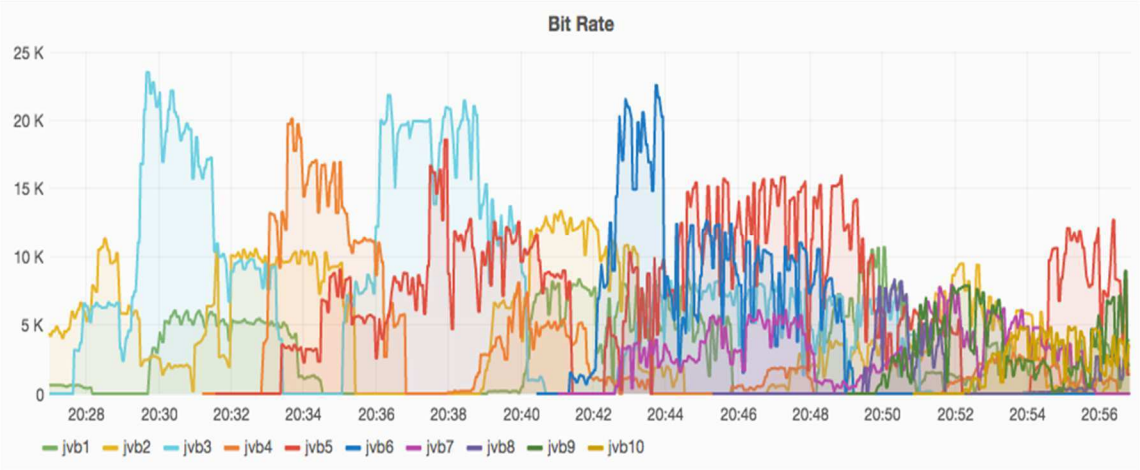
On the other hand, for bootstrapping purposes, we manually start one video bridge and start serving participants. Furthermore, Jitsi Hammer is the load generator for the evaluation. Because it usually requires a lot of memory to simulate hundreds of customers, we used EC2 Spot instances to reduce costs.

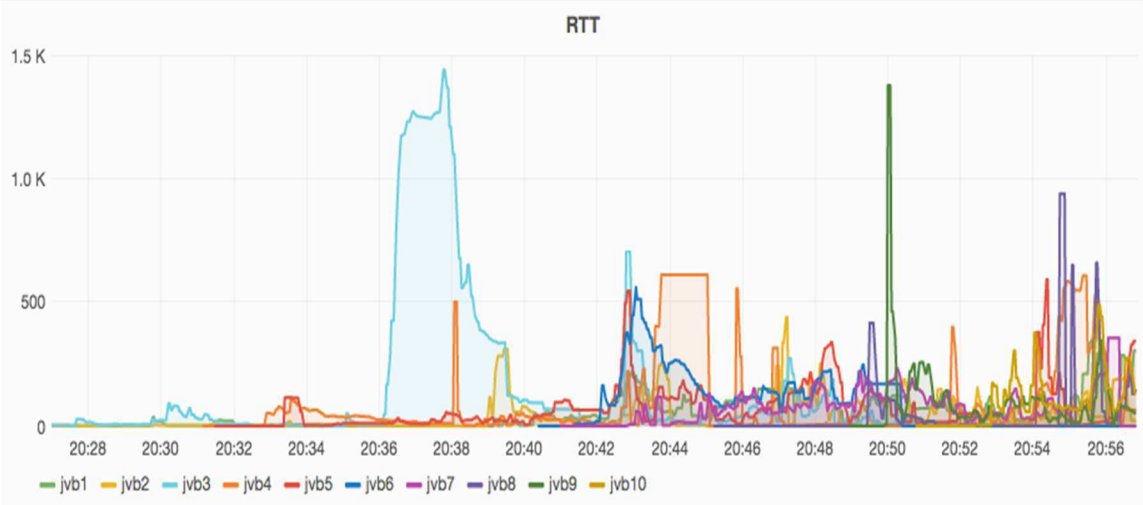
4.2 Horizontal Scaling In EC2 Environment

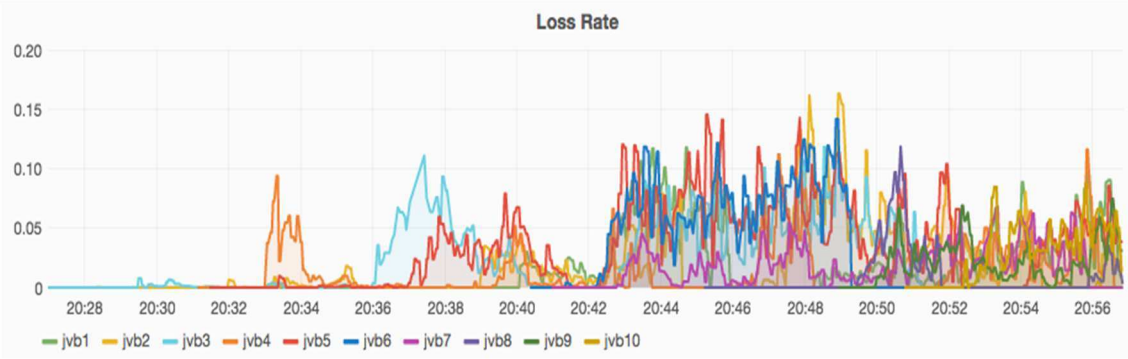
We ran the experiments on three tiers of EC2 instances: t2.micro, t2.small and t2.medium, respectively, and we enabled prediction and scaling components. A single t2.medium Focus server can easily scale up to 40 instances. Below is an example of t2.micro nodes scaled to ten nodes. Due to multiple running instances, the high-availability effect is quite noticeable, which makes the whole system keep serving without much interruption. This horizontal scaling is an example of what happens when Jitsi is run in the cloud to provide quality service.

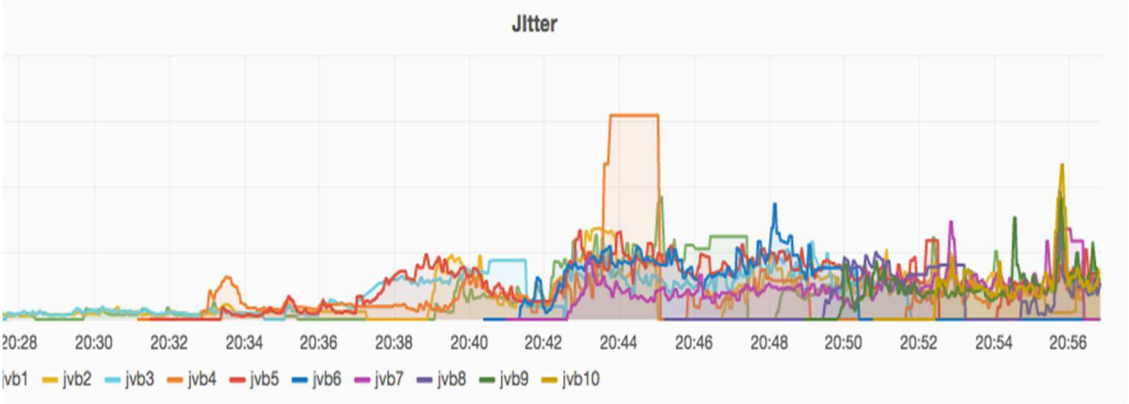
However, the initial pool of servers is customizable, so when the first scaling happens we are sure there are always some servers running. A pool of size one is for demonstration purposes, which helps us see the whole picture during each scaling.

Below are the graphs of the conditions that held when we ran Jitsi on t2.small instances. It shows that new nodes are spawning up as loads are increasing. It also shows many different aspects of the metrics of the Jitsi video-conferencing application.




RTT
1.5 K
1.0 K
500
0
20:28
20:30
20:32
20:34
20:36
20:38
20:40
20:42
20:44
20:46
20:48
20:50
20:52
20:54
20:56
jvb1
jvb2
jvb3
jvb4
jvb5
jvb6
jvb7
jvb8
jvb9
jvb10


Loss Rate
0.20
0.15
0.10
0.05
0
20:28
20:30
20:32
20:34
20:36
20:38
20:40
20:42
20:44
20:46
20:48
20:50
20:52
20:54
20:56
jvb1
jvb2
jvb3
jvb4
jvb5
jvb6
jvb7
jvb8
jvb9
jvb10


Jitter
20:28
20:30
20:32
20:34
20:36
20:38
20:40
20:42
20:44
20:46
20:48
20:50
20:52
20:54
20:56
jvb1
jvb2
jvb3
jvb4
jvb5
jvb6
jvb7
jvb8
jvb9
jvb10

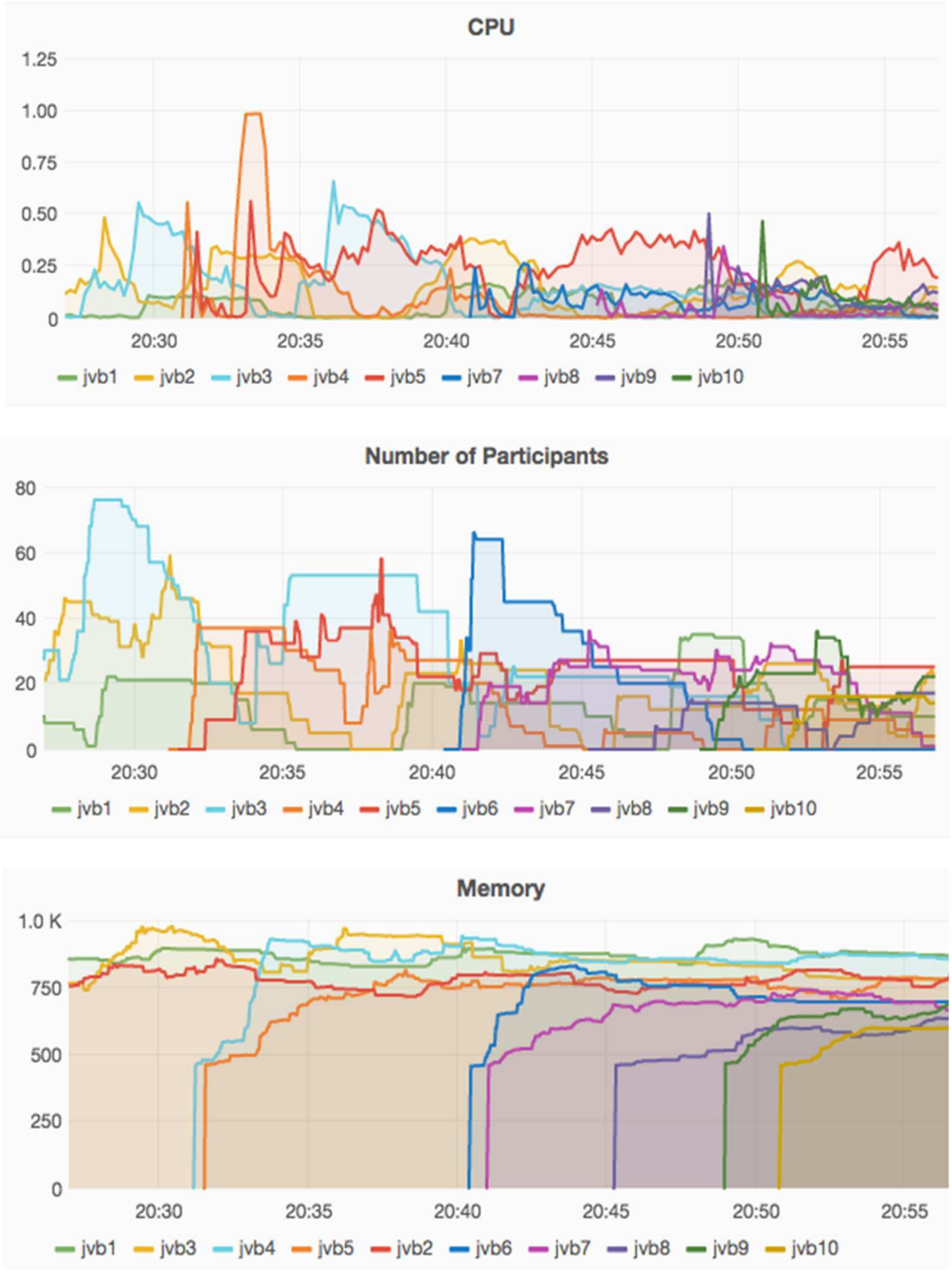


Figure 9- Metrics of Real-time Scaling

We can see from the above graph that, when everything begins, it takes a lot of pressure on the first video bridge. But then our mechanism kicks in, and new servers spawn up to spare loads, and it keeps going in that way. Even though the outcomes of vital metrics are a little

higher than the preset thresholds, everything seems to remain under control without killing any other server in the system.

For example, check the graph of jitter. At the beginning, servers seem to scale accordingly: not many spikes occur during the process. But then some big conferences join the system, and it happens very quickly. The system is then able to sense the peak and quickly spawns up new nodes to compensate. Another example is in the RTT graph: whenever a spike happens, we can assign new conferences to sparer servers. RTT soon returns to normal.

Overall, the system is able to handle continuous loads of conferences and to balance the whole system such that no particular server experiences high loads for a long time. The system runs on an EC2 environment and is capable of doing auto-scaling and load balancing, which directly answers our second research about how to scale in the virtualized environments. One thing to notice is that the metrics continue to reach a little higher than we expected, which is due to the reaction time in the video-conferencing system. Even if we can stop assigning more conferences to the heavily-loaded server, the existing tasks will still contribute to the overall loads. That being said, scaling happens very gracefully. In the next section we show t-tests of the impact after the prediction is introduced.

4.3 Scaling Ahead to Maintain Video-conferencing Quality

To measure the overall statistics, we ran tests with and without the prediction component using five different profiles for all three EC2 instance types. Below is a graph that shows the differences between running with prediction and without. The blue lines are the separator. The results are gathered from a t2.small instance that runs an average profile.

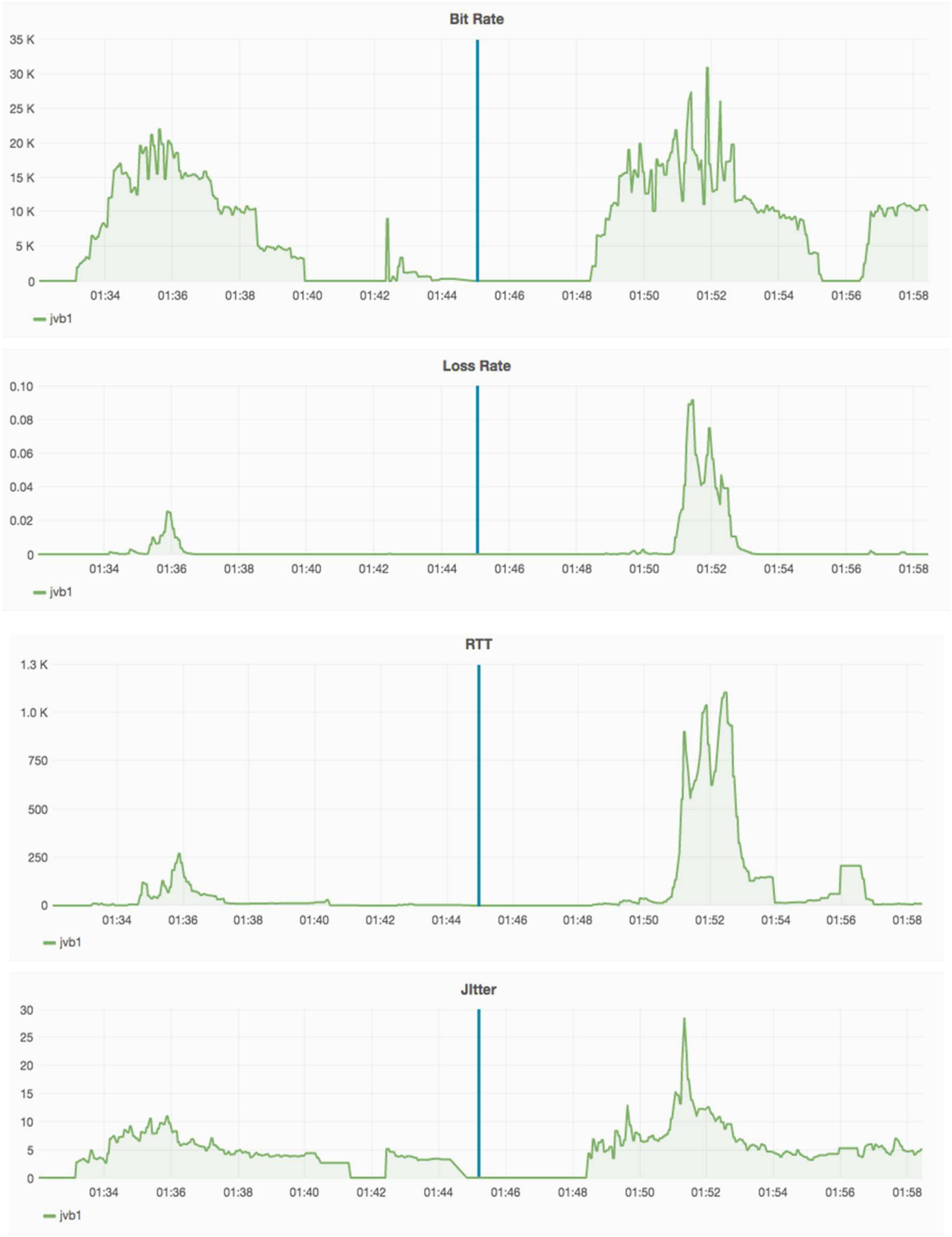


Figure 10- With and Without Prediction

From the above figures, we can observe that the metrics we care about most in videoconferencing, e.g., RTP loss rate, jitters and RTT, are reducing considerably with the prediction component.

On the other hand, we have run t-tests in all three types of EC2 instances and over five workloads. Each workload is running from starting up to reaching thresholds, and finally to stopping. Around 3000 data points are used for the t-test of each instance type. Below are the results of t-tests with 0.05 level of significance. Judging from the p-values, the systems with prediction component tend to have statistically significant lower mean values than the one without it.

Table 7- T-tests After Applying Prediction

EC2 Type	Rtt_aggregate	Jitter_aggregate	Upload loss rate
T2.micro	1.295e-15	6.414e-15	1.538e-06
T2.small	2.2e-16	2.2e-16	1.264e-10
T2.medium	0.02637	0.004017	0.033

Scaling ahead helps us get enough time to start up new servers. At the same time, spikes can be avoided. In our system, there is a customizable variable that enables us to decide how much earlier we can start scaling. Normally, it is set to 5%, which means that we start new nodes 5% before the predicted values reach the thresholds. Since different EC2 instance types can behave slightly differently, we can adjust this value to ensure that high availability is achieved.

Another thing to notice is that smaller servers tend to show more statistically significant results during the comparison. This is because any change in load shows bigger impacts, or in other words, is more sensitive, when a server is provisioned with less CPU and memory. But even for t2.medium, we can still see that the differences are acceptable.

5. CONCLUSIONS AND FUTURE WORK

Given modifications to the Jitsi codebase and the application of prediction models, it is possible to transform Jitsi into a horizontally scalable video-conferencing application in the Amazon Cloud environment. The system we built not only monitors all system conditions; it also maintains quality service by starting new servers rather than by acquiring more powerful servers, which makes operation very elastic and costs much lower.

From experiments, we are able to answer the research questions. For question one, it is desirable to scale ahead of time by applying predictive models. In this way it is possible to avoid exceeding system limits, since virtualized environments are often very sensitive after thresholds are exceeded. The results show noticeable improvements after our models are used for scaling. For question two, we can identify a cost-efficient way to scale in the Amazon Cloud. By setting the right parameters for the initial pool size and time to scale ahead of predicted values, high availability is achieved using the EC2 t2 baseline tiers.

In addition, transiting to the Cloud environment provides the flexibility to easily scale horizontally. The DevOps tools become more and more mature, which eases many pains when it comes to handling a large number of virtual servers.

However, there are still some aspects that we have not covered in thesis. Our work is based mostly at the system-level, with gathered metrics, built models, and applied prediction. To more efficiently scale video-conferencing applications, we must change the session management or even the transport layer, as in [41]. These ideas can be studied in the future. Besides, we have only investigated the Amazon Cloud that is based on XEN technology, but there are many systems that are based on other hypervisors both open source and proprietary. Also, more

training algorithms can be used, such as SVM or Fuzzy Logic, which may better for the non-linear features.

BIBLIOGRAPHY

[1] K. R. Jackson *et al.*, "Performance analysis of high performance computing applications on the amazon web services cloud," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010, pp. 159–168.

[2] W. Lloyd *et al.*, "Demystifying the Clouds: Harnessing Resource Utilization Models for Cost Effective Infrastructure Alternatives," *Appear IEEE Trans. Cloud Comput.*

[3] B. Grozev, L. Marinov, V. Singh, and E. Ivov, "Last n: relevance-based selectivity for forwarding video in multimedia conferences," in *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2015, pp. 19–24.

[4] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications IETF RFC 3550," *U. S.*, 2003.

[5] Y. Okuno, Y. Arai, Y. Hoshi, H. Henmi, T. Otani, and E. Ohba, "XMPP-based energy management system architecture for communications systems," in *Telecommunications Energy Conference (IN•EC), 2015 IEEE International*, 2015, pp. 1–6.

[6] P. Membrey and Y. Demchenko, "Intercloud Control and Management Plane with XMPP," in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, 2015, pp. 471–476.

[7] C. Nesbitt, "Virtual Machine Server Management Tools," Worcester Polytechnic Institute, 2014.

[8] A. Bergkvist, D. Burnett, and C. Jennings, "A. Narayanan," WebRTC 1.0: Real-time Communication Between Browsers," *World Wide Web Consort. WD WD-Webrtc-20120821*, 2012.

[9] M. Wenzel and C. Meinel, “Full-body WebRTC video conferencing in a web-based real-time collaboration system,” in *Computer Supported Cooperative Work in Design (CSCWD), 2016 IEEE 20th International Conference on*, 2016, pp. 334–339.

[10] D. Wing, “Symmetric RTP/RTP Control Protocol (RTCP),” 2007.

[11] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, “An analysis of performance interference effects in virtual environments,” in *2007 IEEE International Symposium on Performance Analysis of Systems & Software*, 2007, pp. 200–209.

[12] W. Lloyd, S. Pallickara, O. David, J. Lyon, M. Arabi, and K. Rojas, “Performance implications of multi-tier application deployments on Infrastructure-as-a-Service clouds: Towards performance modeling,” *Future Gener. Comput. Syst.*, vol. 29, no. 5, pp. 1254–1264, 2013.

[13] S. Kraft, G. Casale, D. Krishnamurthy, D. Greer, and P. Kilpatrick, “IO performance prediction in consolidated virtualized environments,” in *ACM SIGSOFT Software Engineering Notes*, 2011, vol. 36, pp. 295–306.

[14] S. Kundu, R. Rangaswami, K. Dutta, and M. Zhao, “Application performance modeling in a virtualized environment,” in *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, 2010, pp. 1–10.

[15] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta, “Modeling virtualized applications using machine learning techniques,” in *ACM SIGPLAN Notices*, 2012, vol. 47, pp. 3–14.

[16] W. Lloyd, S. Pallickara, O. David, J. Lyon, M. Arabi, and K. Rojas, “Performance modeling to support multi-tier application deployment to infrastructure-as-a-service clouds,” in

Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on, 2012, pp. 73–80.

[17] R. Tibshirani, "Regression shrinkage and selection via the lasso," *J. R. Stat. Soc. Ser. B Methodol.*, pp. 267–288, 1996.

[18] P. Barham *et al.*, "Xen and the art of virtualization," in *ACM SIGOPS Operating Systems Review*, 2003, vol. 37, pp. 164–177.

[19] S. Wind, "Open source cloud computing management platforms: Introduction, comparison, and recommendations for implementation," in *Open Systems (ICOS), 2011 IEEE Conference on*, 2011, pp. 175–179.

[20] S. Kraft, G. Casale, D. Krishnamurthy, D. Greer, and P. Kilpatrick, "Performance models of storage contention in cloud environments," *Softw. Syst. Model.*, vol. 12, no. 4, pp. 681–704, 2013.

[21] G. Casale, S. Kraft, and D. Krishnamurthy, "A model of storage I/O performance interference in virtualized systems," in *2011 31st International Conference on Distributed Computing Systems Workshops*, 2011, pp. 34–39.

[22] K. Ye, X. Jiang, S. Chen, D. Huang, and B. Wang, "Analyzing and modeling the performance in xen-based virtual cluster environment," in *High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on*, 2010, pp. 273–280.

[23] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy, "Profiling and modeling resource usage of virtualized applications," in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, 2008, pp. 366–387.

[24] P. Padala *et al.*, "Automated control of multiple virtualized resources," in *Proceedings of the 4th ACM European conference on Computer systems*, 2009, pp. 13–26.

[25] Z. Gong, X. Gu, and J. Wilkes, “Press: Predictive elastic resource scaling for cloud systems,” in *2010 International Conference on Network and Service Management*, 2010, pp. 9–16.

[26] J. P. Walters *et al.*, “GPU passthrough performance: A comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL applications,” in *2014 IEEE 7th International Conference on Cloud Computing*, 2014, pp. 636–643.

[27] T. Deshane, Z. Shepherd, J. Matthews, M. Ben-Yehuda, A. Shah, and B. Rao, “Quantitative comparison of Xen and KVM,” *Xen Summit Boston MA USA*, pp. 1–2, 2008.

[28] E. C. Amazon, “Amazon elastic compute cloud,” *Retrieved Feb*, vol. 10, 2009.

[29] G. Wang and T. E. Ng, “The impact of virtualization on network performance of amazon ec2 data center,” in *INFOCOM, 2010 Proceedings IEEE*, 2010, pp. 1–9.

[30] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, “A performance analysis of EC2 cloud computing services for scientific computing,” in *International Conference on Cloud Computing*, 2009, pp. 115–131.

[31] J. Dejun, G. Pierre, and C.-H. Chi, “EC2 performance analysis for resource provisioning of service-oriented applications,” in *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, 2010, pp. 197–207.

[32] J. Idziorek, “Discrete event simulation model for analysis of horizontal scaling in the cloud computing model,” in *Simulation Conference (WSC), Proceedings of the 2010 Winter*, 2010, pp. 3004–3014.

[33] V. S. Tiwari, A. Arya, and S. Chaturvedi, “Framework for Horizontal Scaling of Map Matching: Using Map-Reduce,” in *Information Technology (ICIT), 2014 International Conference on*, 2014, pp. 30–34.

[34] A. Naskos *et al.*, “Dependable horizontal scaling based on probabilistic model checking,” in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, 2015, pp. 31–40.

[35] A. Uyar, S. Pallickara, and G. Fox, “Towards an Architecture for Audio/Video Conferencing in Distributed Brokering Systems.,” in *Communications in Computing*, 2003, pp. 17–23.

[36] G. Fox, S. Pallickara, M. Pierce, and H. Gadgil, “Building messaging substrates for Web and Grid applications,” *Philos. Trans. R. Soc. Lond. Math. Phys. Eng. Sci.*, vol. 363, no. 1833, pp. 1757–1773, 2005.

[37] S. Pallickara and G. Fox, “On the Matching of Events in Distributed Brokering Systems.,” in *ITCC (2)*, 2004, pp. 68–76.

[38] G. Fox and S. Pallickara, “Deploying the NaradaBrokering substrate in aiding efficient web and grid service interactions,” *Proc. IEEE*, vol. 93, no. 3, pp. 564–577, 2005.

[39] S. Yi, D. Kondo, and A. Andrzejak, “Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud,” in *2010 IEEE 3rd International Conference on Cloud Computing*, 2010, pp. 236–243.

[40] I.-O. S. T. Series, “Metrics, and Analytics Database,” *Website Httpinfluxdb Com*, 2015.

[41] A. Tarakanov and O. Gushchina, “Use of spatial division scheme in multiple description coding algorithm for multipoint videoconferencing,” in *Control and Communications (SIBCON), 2015 International Siberian Conference on*, 2015, pp. 1–4.