PERFORMANCE MODELLING OF MULTI-TIER CLOUD APPLICATIONS USING SIMPY

by

Dayle Chettiar

Bachelor of Engineering, Electrical and Computer Engineering

University of Mumbai, Mumbai, India, 2010.

A project

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Engineering

in the Program of

Electrical and Computer Engineering.

Toronto, Ontario, Canada, 2016

Author's Declaration

I hereby declare that I am the sole author of this project. This is a true copy of the project report, including any final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this project to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this project by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my project may be made electronically available to the public.

Abstract

PERFORMANCE MODELLING OF MULTI-TIER CLOUD APPLICATIONS USING SIMPY

Dayle J. Chettiar

Master of Engineering (M. Eng.), 2016

Electrical and Computer Engineering, Ryerson University

Todays cloud deployed applications are mostly multi-tiered. Usually, the first tier consists of an Application Service Providers' (ASPs) web servers, the second tier has application servers and the third tier contains database servers. Tiered architectures are often difficult to evaluate in terms of performance. Existing performance models are very effective in finding the mean performance measures. However, metrics such as response-time percentiles are of greater importance to the end-users since it is more desirable to reduce the variability of a system's response time, rather than minimizing the mean response time. In this work, a multi-tier application is modeled as an open queuing network of 3-tiers and the response-time percentiles are estimated using discrete event simulation. Here, we assume that each tier is replicated into a number of copies and each copy runs on a separate Virtual Machine (VM). Although simulation models are computationally more expensive as compared to analytical models, they are much more general. The simulation model of this work can be used as decision support for ASPs in order to determine the optimal configuration of VMs for a given workload such that a required response-time percentile is within a given threshold. In this work, Simpy, a discrete event simulation framework, has been used. The results show that as the number of VMs are increased in a 3-tier open queueing network, the overall system performance (i.e. percentiles and mean response times) does not necessarily become better. The results further show that different system configurations containing the same number of VMs, yield different performance depending on the replication level in different tiers.

Acknowledgments

I would like to acknowledge my family for their continuous support and encouragement, which gave me the ability to pursue my education and other goals.

I would like to thank my supervisor, Dr. Olivia Das, for her support and dedication throughout my graduate experiences and this project. Dr. Das has provided assistance, resources and wisdom throughout the development of this project.

Also, I would like to thank my fellow colleagues and classmates from the Electrical Engineering program at Ryerson University for making my graduate experience unforgettable.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

LIST OF ACRONYMS

1 VM – Virtual Machine

2 DBMS – Database Management System

3 DES – Discrete Event Simulation

4 SLA – Service Level Agreement

5 ASP – Application Service Provider

6 QoS – Quality of Service

7 BPMN – Business Process Modelling Notation

8 UML – Unified Modelling Language

9 QN – Queuing Network

10 WS – Web Server

11 APP – Application Server

12 OS – Operating System

13 DB – Database Server

CHAPTER 1

INTRODUCTION

Application service providers (ASPs), which provide web services to customers have to ensure that their services are always live, ensure minimum delay and adhere to Service Level Agreement (SLA) requirements. ASPs typically buy virtual machines (VMs) from cloud providers in order to cut down their budgets and/ or avoid hosting physical servers at their own premises. Application service providers request cloud service providers for a fixed number of intial VMs to host their applications. Cloud service providers get the VM requirements from ASPs and then bill them i.e. ASPs on a monthly or contract basis [20, 21, 22]. The model described in this research gives application service providers a decision support model in order to forecast how many VMs they should buy, keeping in mind customer Service Level Agreements (SLA) requirements and the workload of jobs that they receive to remain functional [19]. The main objective of this research is to arrive at an appropriate configuration of VMs for a workload of jobs arriving at a multi-tier open queueing system.

Modern cloud deployed applications are mostly multi-tiered. The performance of tiered architectures are often difficult to evaluate. There are two approaches that are commonly used in performance evaluation for computer systems: measurement and modeling. Measurement uses tools to generate an artificial workload to the system and measure its performance. Models, on the other hand, are used to gain an insight into systems that have not been built, or those which are running under conditions such that they are not available for measurements. Performance modeling includes simulation and analytical models. We prefer simulation models, because simulation models are much more general and impose fewer restrictions than

analytical models. However, they are computationally more expensive.

Since resources of application service providers are multi-tiered services, response time is an important statistic, which quantifies the Quality of Service (QoS) of application service providers. Response time generally denotes the amount of time a user waited for a response to a query, without taking into account the quality of the response [11]. Conventionally, the mean response time is an effective QoS metric. However, [14] argues that the response-time percentiles or more generally, the response-time distribution, are of higher importance to the end-users. Although the mean response time of a system may be very low, if we intend to measure performance of a system as the percentage of successful jobs which were processed within a certain response time threshold (percentile), we find that percentiles are a more effective performance criteria. Hence, percentiles (eg 90^{th} or 95^{th} percentiles) are a better QoS metric for end users focussed on performance than mean response time. Along with low response time, end users consuming mission critical applications require that the services that they use are reliable. In this research we will be focussing more on percentiles as a QoS metric rather than average response time since we intend to use a performance metric which ensures that the system's behaviour is consitent rather than faster overall but occasionally inconsistent [14].

In this work, a 3-tier web application is modeled as an open queueing network with three stages with only feed-forward arcs. Here, we assume that each tier is replicated into a number of copies and each copy runs on a separate Virtual Machine (VM) [14]. Thus, the queuing network consists of a variable number of VMs in three stages. The first stage represents web server tier, the second stage represents the application server tier and the third stage represents the database server tier [13]. Each VM is modeled an M/M/1 queue.

Over or under-utilization of VMs could occur if an application service provider didn’t purchase enough number of VMs from the cloud provider for different stages of the open

queueing network to process the incoming jobs. For example, when the number of VMs purchased is too few to handle a given workload, the average response time will be too high and most of the jobs will not be processed within the required response time threshold. On the contrary, if too many VMs were purchased to handle relatively fewer number of jobs, the VMs will be under-utilized and this could lead to wastage of computational resources. Hence, the challenge is to find an optimal configuration for the system consisting of the right number of VMs for each stage to process the incoming jobs that will ensure that a required response-time percentile is within a given threshold.

This work presents a discrete event simulation model to estimate the response time distribution and percentiles. Since it is simulation-based, the model is computationally intensive, however, it is an approximation technique and can accommodate general arrival and service distributions. The rest of the paper is organized as follows. First, background and related work is summarized in chapter 2, along with an introduction to queueing network systems and the types of queueing network systems. A model of the system is given in chapter 3 along with a queueing network model of the system to be simulated. This chapter also presents the simpy [5] simulation framework along with the models used for incoming jobs and VMs. Chapter 4 details the results obtained with the different configurations used for an open queueing network model. Finally, a conclusion with future work is proposed in chapter 5.

CHAPTER 2

BACKGROUND

Application service providers request virtual machines from cloud service providers, for deploying their web applications to satisfy SLA requirements for their own customers. Rather than buying more and more servers as their demand or number of customers increases, application service providers buy virtual machines [20], from cloud service providers like Amazon, Google, Apple, Microsoft, etc.

Application service providers typically can have their resources modeled as a queuing network composed of a set of single-server queues. These single server queues can be thought of as a configuration of 2-3 tiers. Each tier of queues can be visualized as a collection of application or services which are responsible for the execution of requests at each tier. Conventionally, most application service providers have a configuration consisting of 3 tiers where the first tier consists of web servers, the second tier consists of application servers and the third tier consists of DBMS servers [13]. In this fashion, each queue in the single-server queues corresponds to a tier. The configuration can be shown in as in figure 1 below:

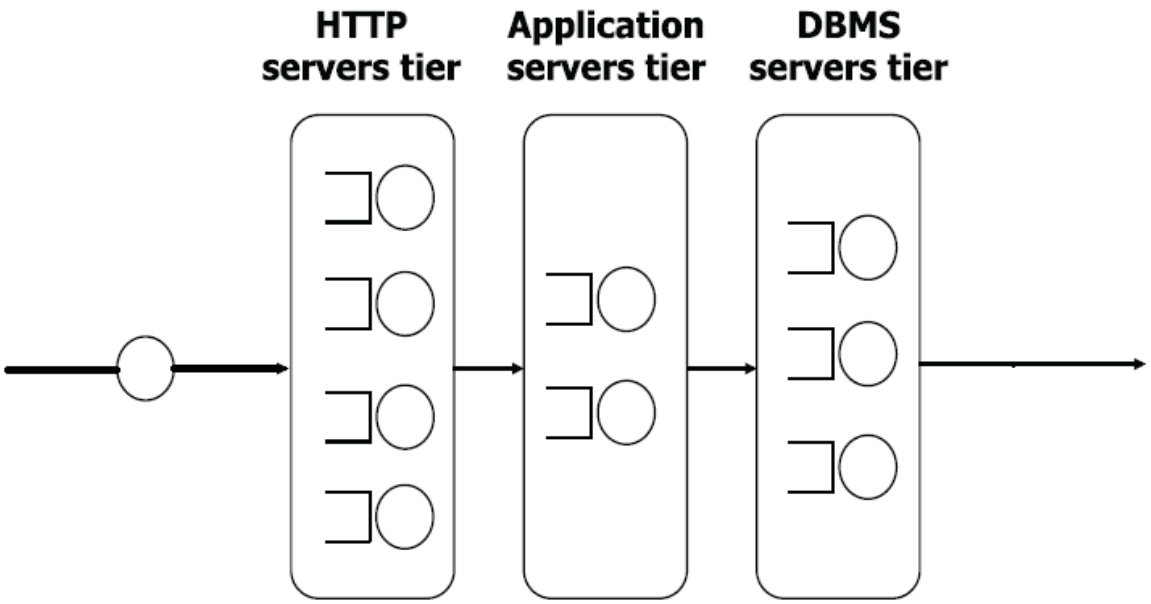

Fig 1. Modern Application Service Provider configuration

Fig. 1 is a sample configuration of an application service provider. The figure above denotes a configuration consisting of 4 VMs at the Web (HTTP) Server tier, 2 VMs at the Application Server tier and 3 VMs at the DBMS server tier. The performance of a system like this can be analyzed using queuing networks, more specifically, open queuing networks. Extensive research has been performed on the problem of computing the response time distribution of queuing networks. Obtaining the response time distribution in a network is generally cumbersome, particularly when a network is not overtake-free [15]. In such cases, approximations are made for response time distributions. An example of such an approximation technique is to divide the network into single server queues. Then, these split queues can be analyzed in isolation [14]. An M/M/c/b queue based approach based on the split queue technique was specified by Woolet [14]. Woolet's technique was further extended by Grottke et al. as in [12] which presents a phase time service distribution. Approximation techniques on obtaining response time distribution for queuing networks with failures of VMs in tiers are presented as in [16] and [12].

Analytical performance models exist for evaluating the performance of systems. Some examples of performance models include Markov chains [1, 2], Petri nets [18, 23, 24], Queueing Networks [1, 2], Layered Queueing Networks (LQN) [17], and the few others. Most of these models can be solved analytically in order to find the mean performance measures. In order to compute mean measures and response time percentiles, we have used simulation models. Simulation models are computationally more expensive as compared to analytical models. However, they are much more general. Simulation models have also been used for other purposes. G. Gagner et al [8] extend the concept of discrete event simulation (DES) towards a UML driven Business Process Modeling Notation (BPMN) for the purpose of simulation modeling. This approach was also used to identify BPMN patterns which were useful for obtaining good simulation models [8]. Pfitzinger et al [9] applied simulation models like Reliability Block Diagrams in order to analyze the efficiency of software and data updates in a mobile distributed electronic toll system. But for the scope of this research, we limit ourselves to the methods presented by [14, 16, 12], where response time distribution for queuing networks was approximated and analyzed. In this research, we are concerned with obtaining the response time distribution of an open queuing network system with replicated servers using discrete event simulation. Similar to R. Paharshingh in [3, 4] who developed a quantitative model for the analysis of hardware, software and response time faults in virtual systems with cloud services, our work is not analytical, but uses simulation using simpy [5]. All the methods presented in this literature review compute the performance of queuing networks using analytical methods, but this research focuses on obtaining performance metrics using simulation.

We first begin with an overview of Discrete Event Simulation in chapter 2.1, then queuing networks systems and the types of queuing network systems are discussed in chapter 2.2.

2.1 DISCRETE EVENT SIMULATION

Using Discrete Event Simulation (DES), the performance of a system can be obtained very accurately. However, it takes a long time to perform the simulation. Also, more resources are needed as compared to analytical models. In DES, it is assumed that no changes occur between successive events in the system to be analysed. The simulation can hop in time from one event to the next. As compared to continuous simulation, DES runs faster since it doesn't have to simulate each and every time slice. Thus, Discrete Event Simulation (DES) approximates any given system as a discrete sequence of events in time [25, 26, 27].

2.2 QUEUING NETWORK SYSTEMS

Queuing network models or systems are those models in which the entire system to be analyzed is represented as a network of queues. A network of queues is a collection of service centers, denoted by nodes. This collection of service centers represents system resources. The arrivals to these service centers can be users, jobs or transactions. Analytic evaluation involves using software to efficiently solve a set of equations induced by the network of queues and its parameters [10].

There are two types of queuing network systems.

2.2.1 OPEN QUEUING NETWORK SYSTEMS

Queueing network systems are called open if the queueing systems can have jobs enter and leave the system. A diagrammatic representation of an open queueing network system is shown in figure 2 [1, 2].

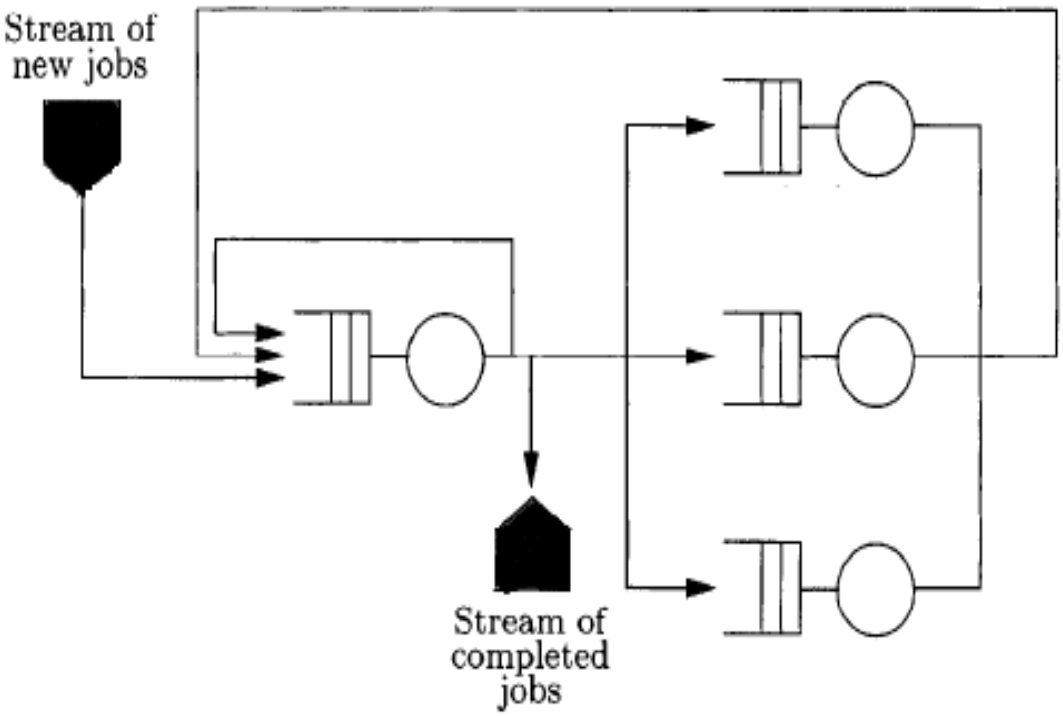

Fig 2: Open Queuing Network System

Thus the number of jobs at any given instance in an open queueing network system is variable, i.e. the number of jobs can increase or decrease. Jobs can enter the queueing system at any node and depart the network from any node [1, 2].

The external arrival rate (λ) of an open queuing network is the sum of the 'arrival rates from outside the system' and the 'arrival rates at all other internal nodes in the system' [1]. The arrival rate of an open queuing network is given in eq. 1 as follows:

$$\lambda_i = \lambda_{0i} + \sum_{j=1}^{N} \lambda_j p_{ji} \qquad for\ i =, \ldots\ldots, N \qquad \text{eq. 1)}$$

where λ_i denotes the arrival rate at the i^{th} node,

λ_{0i} denotes the arrival rate of incoming jobs from outside the system to the i^{th} node,

$\sum_{j=1}^{N} \lambda_j p_{ji}$ denotes the arrival rates to all internal nodes in a system

A Poisson process is a stochastic process which increments the number of events in the system. If the inter-arrival times between each pair of consecutive events is assumed to be independent of other inter-arrival times and the time between a pair of consecutive events has an exponential distribution with arrival rate (λ), then the random variable which denotes the number of incoming jobs has a Poisson distribution [1, 2].

In this research, Poisson arrival rate is used. However, we could also have used a more general arrival rates. When the performance of an open queuing network is to be analysed using simulation, mostly, it is assumed that inter-arrival time distributions are exponential [1].

2.2.2 CLOSED QUEUING NETWORK SYSTEMS

Queuing network systems are called closed if the queuing system has a fixed number of jobs running through the system. Also, the jobs in a closed queuing system can neither enter nor leave the system. A diagrammatic representation of a closed queuing network system is shown in figure 3 [1, 2].

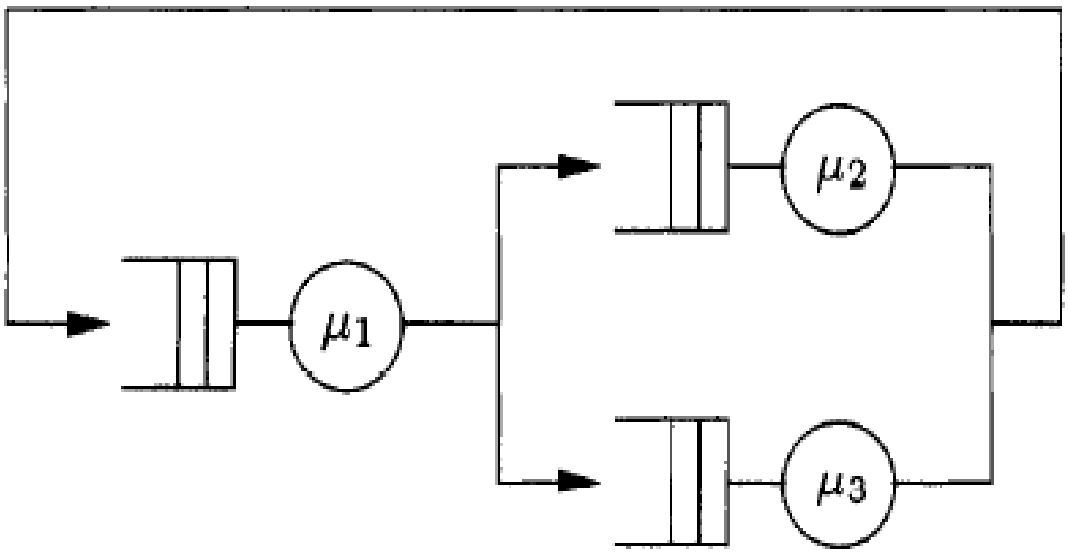

Fig 3. Closed Queuing network

In fig. 3, μ_1, μ_2 and μ_3 are the service rates of each of the system's respective nodes. The number of jobs at all times in a closed queueing network system is fixed, i.e. the same number of jobs is serviced at the nodes in the closed model. A fixed number of jobs are initialized in the closed queueing system once and then those jobs get circulated throughout the closed queueing network and cannot leave the system [1, 2]. A major disadvantage in closed queuing networks is that while modeling the system, the designer is supposed to know the number of jobs that will enter the system in advance.

The external arrival rate (λ) of a closed queuing network is the sum of all the 'arrival rates from all the nodes in the system'. The arrival rate of a closed queuing network is given in eq. 2 as follows:

$$\lambda_i = \sum_{j=1}^{N} \lambda_j p_{ji} \qquad for\ i =, \ldots\ldots, N \qquad \text{eq. 2)}$$

since no jobs enter the closed system externally.

where

$\sum_{j=1}^{N} \lambda_j p_{ji}$ denotes the arrival rates to all nodes in a closed queuing network.

Now, any queueing network system can be solved either analytically or using discrete event simulation. The methods presented in the literature have all discussed analytical models, but in this research, we are use simulation to obtain performance of the queuing network system.

CHAPTER 3

MODEL

The application architecture to be analyzed in this research is given in figure 4.

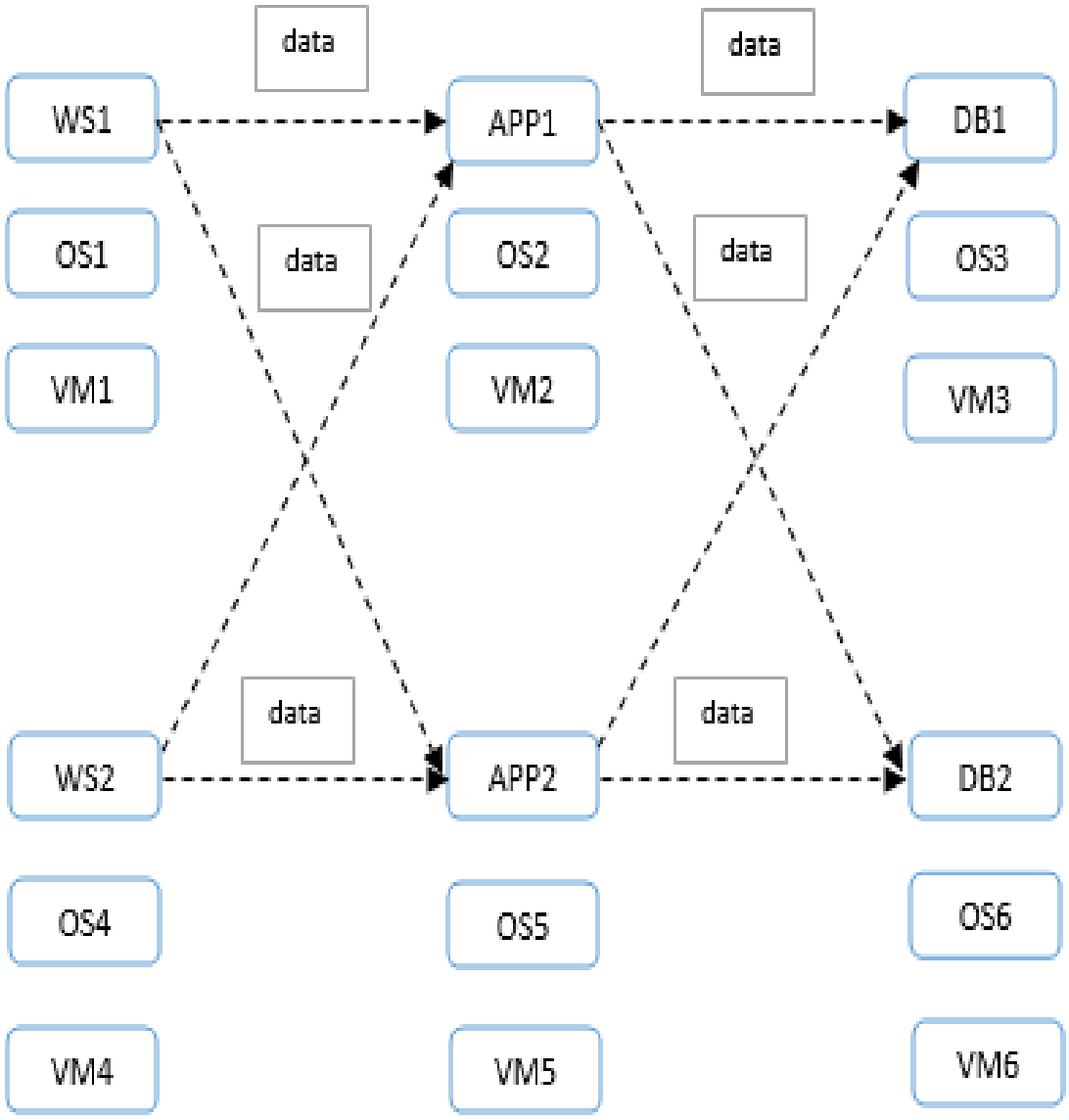

Fig 4. Hardware and software virtual system

Figure 4 shows the architecture of a web app consisting of 3 tiers or servers. The first tier consists of web servers, the second tier consists of application servers and the third tier consists of database servers. From the fig. 4, there is a load balancer between the 3 tiers. The load, from Web Server 1 (WS1) will be equally distributed between Application Server 1 (APP1) and Application Server 2 (APP2). Similarly, the load from Web Server 2 (WS2) will be distributed between APP1 and APP2. In the next stage, the load from Application Server 1 (APP1) will be equally distributed between Database Server 1 (DB1) and Database Server 2 (DB2). The load from Application Server 2 (APP2) will be distributed between DB1 and DB2. Also note that WS1 runs on top of Operating System 1 (OS1), which in turn runs on top of Virtual Machine 1 (VM1). Similarly, APP1 and DB1 run on top of Operating System 2 (OS2), Virtual Machine 2 (VM2) and Operating System 3 (OS3), Virtual Machine 3 (VM3) respectively.

Identically, WS2, APP2 and DB2 run on Operating System 4 (OS4), Virtual Machine 4 (VM4) and Operating System 5 (OS5), Virtual Machine 5 (VM5) and Operating System 6 (OS6), Virtual Machine 6 (VM6) respectively similar to [3].

We assume that each tier is replicated into a number of copies and each copy runs on a separate Virtual Machine (VM) [14].

The above fig. 4 is an example configuration of (2, 2, 2). That is, 2 web servers in the first tier, followed by 2 application servers in the second tier and finally 2 database servers in the third tier. The model in figure 4 can have different configurations with varying number of servers (i.e. a configuration of (1, 1, 1) to (3, 3, 3)) in each of the three tiers depending on the workload of incoming jobs.

3.1 QUEUING NETWORK MODEL

The figure 4 can be modeled as an open queuing network, when resources, WS1 on VM1, APP1 on VM2, DB1 on VM3, WS2 on VM4, APP2 on VM5 and DB2 on VM6 are all operational as given in figure 5. Each VM is modelled as an M/ M/ 1 queue in this research. In this system, all jobs coming into the system will go out of it. Also, all incoming jobs have an exponential service time and their inter-arrival time is exponential.

In addition to that, note that there is a load-balancer between all 3 tiers in the replica, i.e. depending on the number of VMs in each tier, the load-balancer will balance the workload of jobs. Thus, the queuing network model for figure 4 is given in figure 5. Note that 'λ' denotes arrival rate and 'μ' denotes the service rate.

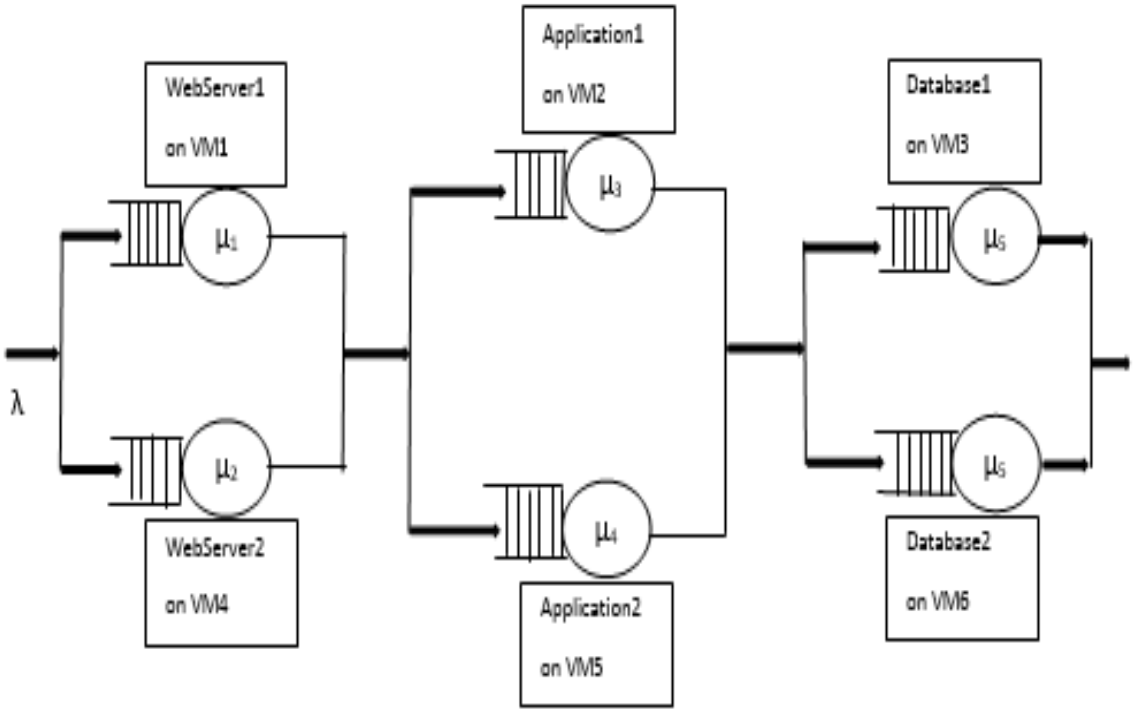

Fig 5. Queuing Network Model

In the scope of this research, both web servers, (WS1 and WS2), both application servers, (APP1 and APP2) and both database servers (DB1 and DB2) are operational. This

configuration is shown in the queuing network in Figure 5.

The fig. 5 is an example configuration of (2, 2, 2). That is, 2 VMs in the first tier, followed by 2 VMs in the second tier and finally 2 VMs in the third tier. The model in fig. 5 can have a range of different configurations with varying number of servers (i.e. a configuration of (1, 1, 1) to (3, 3, 3)) in each of the three tiers.

In the system, requests enter at a rate of 'λ' and are distributed equally in the first stage, among VM1 and VM4. After the requests are processed after the first stage, they are again distributed equally among VM2 or VM5. Finally, in the third stage, the requests are distributed between VM3 and VM6. Hence, it can be seen that the load is equally distributed at each stage of the multi-tiered system and the routing probabilities are determined based on how many servers there are in each tier.

This system in figure 5 can be simulated for performance to predict the number of VMs the application service provider will require at each of the 3 tiers (Web Server, Application Server and DBMS server) for a workload of jobs as discussed previously using DES. The application service provider is provided a decision support system to predict how many VMs he/ she will need to buy for a given workload of jobs. Also care has been taken to satisfy percentile requirements and response time requirements such that the workload of jobs arriving at the 3 tier queueing network configuration satisfies the response time requirement (in seconds).

3.2 SIMPY

The framework

The 'Simpy' framework is a process based, discrete event simulation framework based on the python programming language. The latest version of the simpy simulation framework is

3.0.8 and can be found at [5]. Processes in simpy can be written using python generator functions. Some example scenarios that can be analyzed for performance using simpy are:

i) A number of cars arrive at a car wash shop. We are interested in finding out the optimal number of car wash shops for a given workload. Also, the average response time for a car to get washed in this system can be obtained using simulation [5].

ii) A number of customers arriving at multiple or a single cashiers of a fast food restaurant.

iii) A number of students arriving at the borrowing checkout of a library.

Resources are an important concept in simpy. Resources are a kind of container with limited capacity. If a resource is full or empty, they are supposed to enter a queue and wait for a signal [5]. Resources can be monitored for particular events to occur for future analysis. Example of resources are 'jobs', 'running processes' or even the 'time taken by a specific process to complete'. The monitored resources can then be used to simulate queueing network systems in order to obtain performance metrics like average response time, number of jobs completed successfully, throughput, proportion of jobs completed within a certain time frame as compared to total number of jobs in the system. Simulations in simpy can be performed for a certain specific time interval or infinity (ideally till all jobs are serviced) [5, 6, 7].

The function, 'environment.process()' can be used to start a process. The 'Process' instance in simpy is used for process interaction with other processes or models. Examples of process interactions are

→ waiting for another interacting process to finish execution

→ interrupting another process while it is waiting for another event.

Also, 'Timeout' is an event that allows a process to sleep for a given amount of time. The 'Timeout' event type is triggered after a certain simulation time or time threshold has passed. The function, 'interrupt()' can be used to interrupt a running process. Any process or a set of

processes can be set to 'monitored=true' as a parameter to monitor the process in order to analyze the process' performance [5, 6, 7].

3.2.1 MODELS AND PROCESS SETUP

The model for a 'Source' is given in figure 6.

```
class Source(Process):
    """ Source generates jobs randomly"""
    def __init__(self,sys):
        Process.__init__(self)
        self.sys = sys

    def generate(self, number, arrRate, processors):
        for i in range(number):
            c = Job("Job%02d" % (i), i, self.sys)
            activate(c, c.visit(processors))
            t = expovariate(arrRate)
            yield hold, self, t
```

Fig 6. Source process

In this pseudocode, the class 'Source' generates jobs randomly. This is done in the Source class' '`generate()`' function. The `number`=1000, which indicates that there will be 1000 jobs in the system. '`arrRate`' denotes the arrival rate of jobs from outside (source) to the open queueing network model. '`processors`' denotes the number of VMs at any one of the 3 tiers currently under consideration. The code which follows assigns jobs to the configured VMs in the tiers. Also, each of these jobs are generated exponentially with an interval of '`expovariate(arrRate)`', where `arrRate` is the arrival rate of incoming jobs. '`__init__()`' is a constructor for the 'Source' class which initializes default parameter values

for the 'Source' class.

The model for an incoming 'job' is given in figure 7.

```
class Job(Process):
    """ Job arrives, chooses the shortest queue is served and leaves """
    def __init__(self,name,i,sys):
        Process.__init__(self)
        self.name=name
        self.i=i
        self.sys = sys

    def visit(self, processors):
        arrivalToSystem = now()
        print ("%8.5f %s: Arrives     "%(now(),self.name))

        arrive = now()
        ##Job then moves to first level of servers
        p = numberOfL1VMs
        j = random.randint(1, p)
        yield request,self,processors[j-1]
        wait = float(now()-arrive)
        print ("%3.8f %s: Waited for %3.8f for %s"
               %(now(),self.name,wait, processors[j-1].name))
        tiw = expovariate(serviceRateL1)
        yield hold,self,tiw
        yield release,self,processors[j-1]
```

Fig 7. Model for an arriving job.

In this code, '`__init__()`' is a constructor for the 'Job' class which initializes default parameter values for the 'Job' class. In this class, a job arrives, chooses the shortest queue, gets served and leaves. This is done in the '`visit()`' function. The arrival instant of each job is

recorded and displayed as in statement '`arrivalToSystem=now()`'. The parameter '`numberOfL1VMs`' denotes the number of VMs to be used for simulation at the first tier. The python generator, '`yield request`' is used to request a processor for an arriving job. The wait time between job arrival and the instant at which requested processor was assigned to a specific job is given by the statement '`wait = float(now() - arrive)`'. The time in waiting, '`tiw`' is exponential. The generator, '`yield hold`' can be used to service the job for an exponential time. Then the generator, '`yield release`' can be used to release the servicing of the job by a processor. Note that the above configuration for an arriving job is depicted for the first tier only as in figure 5. For the remaining two tiers, you need to have to repeat the pseudocode for parameters '`numberOfL2VMs`' and '`numberOfL3VMs`', which denote the number of VMs in the second and third tiers respectively. For a full reference of the 3 tier queuing network model refer the code appendix [appendix A].

CHAPTER 4

RESULTS

The initial parameters for performance of this queueing network model is given as in figure 8.

```
## Parameters ------------------------

maxNumber = 1000
endTime = 2000.0     # seconds
serviceRateL1 = 60   # 60 jobs per second
serviceRateL2 = 70   # 70 jobs per second
serviceRateL3 = 80   # 80 jobs per second
arrRate = 100      # 100 arrivals per second
nrRuns = 3        # number of simulation runs
theseed = 787878
processors = []

responseTimeReq = 0.7
numberOfL1VMs = 3
numberOfL2VMs = 3
numberOfL3VMs = 3
```

Fig 8. Initial parameters

From fig. 8, the system is simulated for 1000 jobs and for 2000 seconds which is approximately 33 minutes. The arrival rate of jobs, '`arrRate`' into the queueing network system is 100 jobs/second. The service rate of the jobs arriving at the first, second and third tier in the queueing network model is set to 60, 70 and 80 jobs per second respectively. There are 3 VMs ('`numberOfL1VMs`', '`numberOfL2VMs`' and '`numberOfL3VMs`') each, in the first, second and third tier of the system. For our research, we consider a response time threshold

range between 0.2 to 0.7 seconds. For the purpose of this run, this threshold is set to '`responseTimeReq=0.7`' where `responseTimeReq` denotes the response time requirement or threshold in order to compute percentiles**.** In order to ensure consistency and completeness, the system is simulated for 3 runs (as given by `nrRuns`) and the average of the three runs is obtained as the system performance. In this analysis, we use a configuration consisting of varying number of resources, depending on the number of VMs needed at each of the three tiers as in figure 5.

Using the initial parameters as in fig. 8, we can obtain performance metrics for the queueing network model described in figure 5. This is shown in figure 9.

```
class Job(Process):
    """ Job arrives, chooses the shortest queue
        is served and leaves
    """
    def __init__(self,name,i,sys):
        ..........................................................
        ..........................................................
        ................Job class' constructor.................
        ..........................................................
        ..........................................................

    def visit(self, processors):
        arrivalToSystem = now()
        print ("%8.5f %s: Arrives     "%(now(),self.name))

        ..........................................................
        ..........................................................
        .....first stage model for jobs as in figure7............
        ..........................................................

        ..........................................................
        ..........................................................
        .....second stage model for jobs as in figure7...........
        ..........................................................

        ..........................................................
        ..........................................................
        .....third stage model for jobs as in figure7............
        ..........................................................


        ## Job processing is done and the job leaves the network
        print ("%8.5f %s: Finished      "%(now(),self.name))
        responseTime = now() - arrivalToSystem
        print("%8.5f %s Response time: %2f" % (now(), self.name, responseTime))
        if responseTime <= responseTimeReq:
            self.sys.numberMeetDeadline = self.sys.numberMeetDeadline + 1
        self.sys.wm1.observe(responseTime)
```

Fig 9. Performance measurement

In fig. 9, '`arrivalToSystem`' is the instant at which a job arrives for getting serviced by a VM. Fig. 7 gave a model for a single tier. The model can be modified accordingly as in for different number of VMs in its tiers as in figure 9 in the '`visit()`' module. '`responseTime`'

is a performance metric for obtaining the response time of a job for getting itself processed. It is the difference between the current time and the time marked by the job's '`arrive`' parameter. Also, from fig. 8, if any of the jobs' response time is lesser than the `responseTimeReq` for that iteration, then a counter, '`numberMeetDeadline`' is incremented. The '`numberMeetDeadline`' is used incrementally in order to compute the percentile of the current VM configuration. The statement '`observe(responseTime)`' is used to monitor future jobs which meet the threshold.

The figure 10 below gives the queueing network model as a whole.

```
## Model components ----------------------
class QN():
  def model(self, nrRuns, num, arrRate, endTime):
    averageRespTimeSum = 0.0
    probMetDeadlineSum = 0.0
    for runNr in range(nrRuns):
      self.wm1 = Monitor(name='ResponseTime')
      self.numberMeetDeadline = 0

      initialize()
      source = Source(self)
      activate(source,source.generate(num, arrRate, processors), at=0.0)
      simulate(until=endTime)

      if(self.wm1.count() > 0):
        result1 = self.wm1.count(), self.wm1.mean(), self.wm1.total()
        averageRespTimeSum = averageRespTimeSum + self.wm1.mean()
        print("Average response time for %3d completions was %3.3f seconds.
            Total response time = %3.5f." % result1)
        probMetDeadline = float(self.numberMeetDeadline)/self.wm1.count()
        probMetDeadlineSum = probMetDeadlineSum + probMetDeadline
        print("%d out of %d Jobs met the response time requirement of %f"
            % (self.numberMeetDeadline, self.wm1.count(), responseTimeReq))
        print("Probability of Jobs that met the response time requirement of %3.2f is %3.3f"
            % (responseTimeReq, probMetDeadline))
        print ("%s run(s) completed" %(runNr + 1))

    print("*****************************************************************************")
    print("Configuration is (%d,%d,%d)" % (numberOfL1VMs,numberOfL2VMs,numberOfL3VMs))
    print("Average response time over %d runs is %3.3f seconds."
        % (nrRuns, averageRespTimeSum/nrRuns))
    print("Probability of Jobs that met the response time requirement
        of %3.2f over %d runs is %3.3f" % (responseTimeReq, nrRuns, float(probMetDeadlineSum)/nrRuns))
    print("*****************************************************************************")
```

Fig 10. Open queueing network model

This model is initialized using the statement '`initialize()`'. It generates 1000 random jobs using the statement '`source = Source(self)`'. The jobs are activated and arrive at

VMs using the statement

```
'activate(source,source.generate(num, arrInt, processors) at = 0.0) '
```

Between two arrivals, the time is 1 / 100 seconds since `arrTime` = 100. The result is obtained as the number of jobs processed, average response time and total response time.

The final 2 statements in fig. 10 gives the 'average response time over 3 runs' and the probability that the jobs meet the threshold limit to get processed. This threshold limit can lie between the range of 0.2 to 0.7 seconds.

There can be 27 different configurations for fig 9 and 10 if we modify the '`numberOfL1VMs`', '`numberOfL2VMs`' and '`numberOfL3VMs`' parameters accordingly in fig. 8, between (1, 1, 1) and (3, 3, 3).

The above configuration was for the system as in figure 5, consisting of 3 tiers which could have (1, 2, or 3) VMs at the first tier, (1, 2, or 3) VMs at the second tier and (1, 2, or 3) VMs at the third tier. The configuration can be modelled as having greater than 3 VMs at any tier. However, the intention is to find out whether we get a probability that the jobs meet the threshold limit of (0.2 to 0.7) seconds for a workload of 1000 jobs, which is shown to increase as the number of VMs is increased in the first stage.

Now, the average response time is given by,

$$Average\ response\ time = \ Mean\ response\ time,$$

$$Percentile = \frac{Number\ of\ jobs\ completed}{Total\ number\ of\ jobs} \quad within\ a\ given\ response\ time\ range$$

So, for

arrival rate(λ) = 100 jobs/second,

service rate of tier 1 (μ_1) = 60 jobs/second,

service rate of tier 2 (μ_2) = 70 jobs/second,

service rate of tier 3 (μ_3) = 80jobs/second

and a response time requirement between 0.2 to 0.7 seconds. The code in figure 8, 9 and 10 were modelled to have varying number of VMs and the following results were obtained.

For this research we assume that a percentile of greater than 70% is ideal and a percentile of less than 50% is wasteful since that would mean that only 50% of jobs satisfy a given response time requirement, which is not ideal. A table of results for average response times for different number of VMs in the three tier configuration is given in the table 1.

Configuration	(1, 1, 1)	(1, 2, 1)	(1, 1, 2)	(2, 1, 1)	(1, 1, 3)	(1, 3, 1)	(3, 1, 1)	(2, 2, 1)	(2, 1, 2)
Avg response time	3.517	3.455	3.334	2.155	3.433	3.540	2.878	1.451	2.421
Configuration	(1, 2, 2)	(2, 2, 2)	(1, 2, 3)	(1, 3, 2)	(2, 1, 3)	(2, 3, 1)	(3, 2, 1)	(3, 1, 2)	(3, 3, 1)
Avg response time	3.376	0.155	3.395	3.096	2.035	1.237	1.421	2.459	1.230
Configuration	(3, 1, 3)	(1, 3, 3)	(2, 3, 2)	(2, 2, 3)	(3, 2, 2)	(3, 3, 2)	(3, 2, 3)	(2, 3, 3)	(3, 3, 3)
Avg response time	2.314	3.057	0.134	0.144	0.117	0.098	0.108	0.162	0.093

Table 1: Average response times for VM configurations ranging between (1,1,1) to (3,3,3)

From table 1, it can be seen that as the sum of the number of VMs, increases in the three tiers of the configuration as in figure 5, in general, the average response time for processing all jobs decreases. For example, the average response time of the (1, 1, 1) configuration (sum = 3) is 3.517, while the average response time of the (2, 1, 1) configuration (sum = 4) is 2.155.

However, note the (2, 2, 2) configuration (sum = 6) which has a much lower mean response time than some configurations with higher sums of VMs (i.e. (1, 3, 3), (3, 1, 3) and (2, 3, 3)). In this case, when the (1, 3, 3) and (3, 1, 3) configurations are compared to the (2, 2, 2) configuration, there are a fewer number (i.e. in this case just 1) of VMs in either tiers 1 or 2 respectively which process incoming jobs as compared to the (2, 2, 2) VM configuration. Hence, just the one VM in (1, 3, 3) and (3, 1, 3) configuration acts as a bottleneck for processing incoming jobs. Having a comparatively less number of VMs or servers in the configuration of a multi-tier open queuing network systems is a bottleneck in the system. Thus, even if an ASP buys more VMs, it is not guaranteed that the mean response time of the configuration will be better.

Also in table 1, notice that for different configurations with the same sum of number of VMs, (i. e. For (2, 3, 3), (3, 2, 3), or (3, 3, 2) where sum of VMs is 8), the VM configuration which has a higher number of VMs in the first tier has a better average response time, followed by the VM configuration that has more VMs in the second tier and so on.

The current configuration has service rate of tier 1 (μ_1) = 60 jobs/second, service rate of tier 2 (μ_2) = 70 jobs/second and service rate of tier 3 (μ_3) = 80 jobs/second. i.e. $\mu_1 < \mu_2 < \mu_3$. Depending on the arrival rate (λ), the mean response time will vary if the service rate of tier 1 (μ_1) is higher than that of tier 2 (μ_2) and tier 3 (μ_3). This would just mean that jobs were processed at a faster rate in tier 1 than tiers 2 and 3. Hence, more jobs will be processed by tier 1 than tiers 2 and 3. Thus, the mean response time will be lower if $\mu_1 > \mu_2 > \mu_3$.

Results after 3 runs										
	Response time requirement									
Configuration		(1, 1, 1)	(1, 2, 1)	(1, 1, 2)	(2, 1, 1)	(1, 1, 3)	(1, 3, 1)	(3, 1, 1)	(2, 2, 1)	(2, 1, 2)
	0.200	0.022	0.028	0.025	0.028	0.033	0.028	0.024	0.051	0.043
	0.300	0.035	0.043	0.040	0.054	0.044	0.043	0.033	0.078	0.061
Percentiles	0.400	0.045	0.058	0.046	0.081	0.058	0.055	0.051	0.092	0.084
	0.500	0.056	0.072	0.056	0.101	0.071	0.068	0.069	0.149	0.095
	0.600	0.072	0.081	0.074	0.128	0.084	0.088	0.084	0.191	0.109
	0.700	0.085	0.090	0.087	0.149	0.093	0.109	0.097	0.235	0.125
Configuration		(1, 2, 2)	(2, 2, 2)	(1, 2, 3)	(1, 3, 2)	(2, 1, 3)	(2, 3, 1)	(3, 2, 1)	(3, 1, 2)	(3, 3, 1)
	0.200	0.038	0.754	0.027	0.032	0.025	0.123	0.065	0.042	0.087
	0.300	0.050	0.927	0.039	0.046	0.048	0.184	0.110	0.060	0.120
Percentiles	0.400	0.066	0.972	0.055	0.058	0.074	0.223	0.156	0.076	0.189
	0.500	0.090	0.989	0.064	0.068	0.118	0.246	0.200	0.089	0.261
	0.600	0.104	0.997	0.089	0.079	0.152	0.266	0.227	0.107	0.319
	0.700	0.120	1.000	0.113	0.091	0.193	0.281	0.241	0.125	0.350
Configuration		(3, 1, 3)	(1, 3, 3)	(2, 3, 2)	(2, 2, 3)	(3, 2, 2)	(3, 3, 2)	(3, 2, 3)	(2, 3, 3)	(3, 3, 3)
	0.200	0.036	0.045	0.809	0.767	0.887	0.941	0.899	0.717	0.941
	0.300	0.054	0.061	0.961	0.946	0.994	0.994	0.988	0.895	0.994
Percentiles	0.400	0.065	0.080	0.994	0.995	1.000	1.000	0.999	0.965	1.000
	0.500	0.097	0.105	0.998	1.000	1.000	1.000	1.000	0.983	1.000
	0.600	0.141	0.116	1.000	1.000	1.000	1.000	1.000	0.995	1.000
	0.700	0.159	0.131	1.000	1.000	1.000	1.000	1.000	1.000	1.000

Table 2: Percentiles for different VM configurations between (1,1,1) to (3,3,3)

A table of results for percentiles for different number of VMs in the three tier configuration is given in the table 2. Note that a majority of the configurations have a percentile of less than 50%. Since we assume that a percentile of greater than 70% is ideal and a percentile of less than 50% is wasteful, table 3 limits VM configurations to (2, 2, 2), (2, 3, 2), (2, 2, 3), (3, 2, 2), (3, 3, 2), (3, 2, 3), (2, 3, 3) and (3, 3, 3) since all other VM configurations have a percentile of less than 70%. This table also takes into account the response time requirement or threshold between the range of 0.2 seconds to 0.7 seconds. Table 3 is basically a condensed version of table 2, limited to percentiles greater than 0.7 (i.e. 70%).

Results after 3 runs									
	Response time requirement								
Configuration		(2, 2, 2)	(2, 3, 2)	(2, 2, 3)	(3, 2, 2)	(3, 3, 2)	(3, 2, 3)	(2, 3, 3)	(3, 3, 3)
	0.200	0.754	0.809	0.767	0.887	0.941	0.899	0.717	0.941
	0.300	0.927	0.961	0.946	0.994	0.994	0.988	0.895	0.994
Percentiles	0.400	0.972	0.994	0.995	1.000	1.000	0.999	0.965	1.000
	0.500	0.989	0.998	1.000	1.000	1.000	1.000	0.983	1.000
	0.600	0.997	1.000	1.000	1.000	1.000	1.000	0.995	1.000
	0.700	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

Table 3: Percentiles for VM configurations which satisfy a performance criteria

From table 2 and 3, it can be seen that as the sum of the number of VMs, increases in the three tiers of the configuration as in figure 5, in general, the percentiles for processing all jobs increases for each of the given thresholds. For example, the percentile of the (1, 1, 1) configuration (sum = 3) and `responseTimeReq` = 0.4 seconds is 0.045, while the percentile of the (2, 1, 1) configuration (sum = 4) and `responseTimeReq` = 0.4 seconds is 0.081. These two configurations can be checked for the response time requirement range (0.2 to 0.7).

However, note the (2, 2, 2) configuration (sum = 6) as in table 2, which has a much better percentile than some configurations with higher sums of VMs (i.e. (1, 3, 3), (3, 1, 3) and (2, 3, 3)) for different response time requirements. In this case, when the (1, 3, 3) and (3, 1, 3) configurations are compared to the (2, 2, 2) configuration, there are a fewer number (i.e. in this case just 1) of VMs in either tiers 1 or 2 respectively which process incoming jobs as compared to the (2, 2, 2) VM configuration. Hence, just the one VM in (1, 3, 3) and (3, 1, 3) configuration acts as a bottleneck for processing incoming jobs. Having a comparatively less number of VMs or servers in the configuration of a multi-tier open queuing network systems is a bottleneck in the system. Thus, even if an ASP buys more VMs, it is not guaranteed that the percentiles will be better.

Also notice that for different configurations with the same sum of number of VMs, (i. e. for (2, 3, 3), (3, 2, 3), or (3, 3, 2) where sum of VMs is 8), the VM configuration which has a higher number of VMs in the first tier has a better percentile, followed by the VM configuration that has more VMs in the second tier and so on.

Similar to what was found with mean response time, depending on the arrival rate (λ), the percentiles will vary if the service rate of tier 1 (μ_1) is higher than that of tier 2 (μ_2) and tier 3 (μ_3). This would just mean that jobs were processed at a faster rate in tier 1 than tiers 2 and 3. Hence, more jobs will be processed by tier 1 than tiers 2 and 3. Thus, the percentiles will be greater if $\mu_1 > \mu_2 > \mu_3$.

A graph plotting the average response time for different VM configurations in the three tiers is shown separately in figures 11, 12 and 13 below:

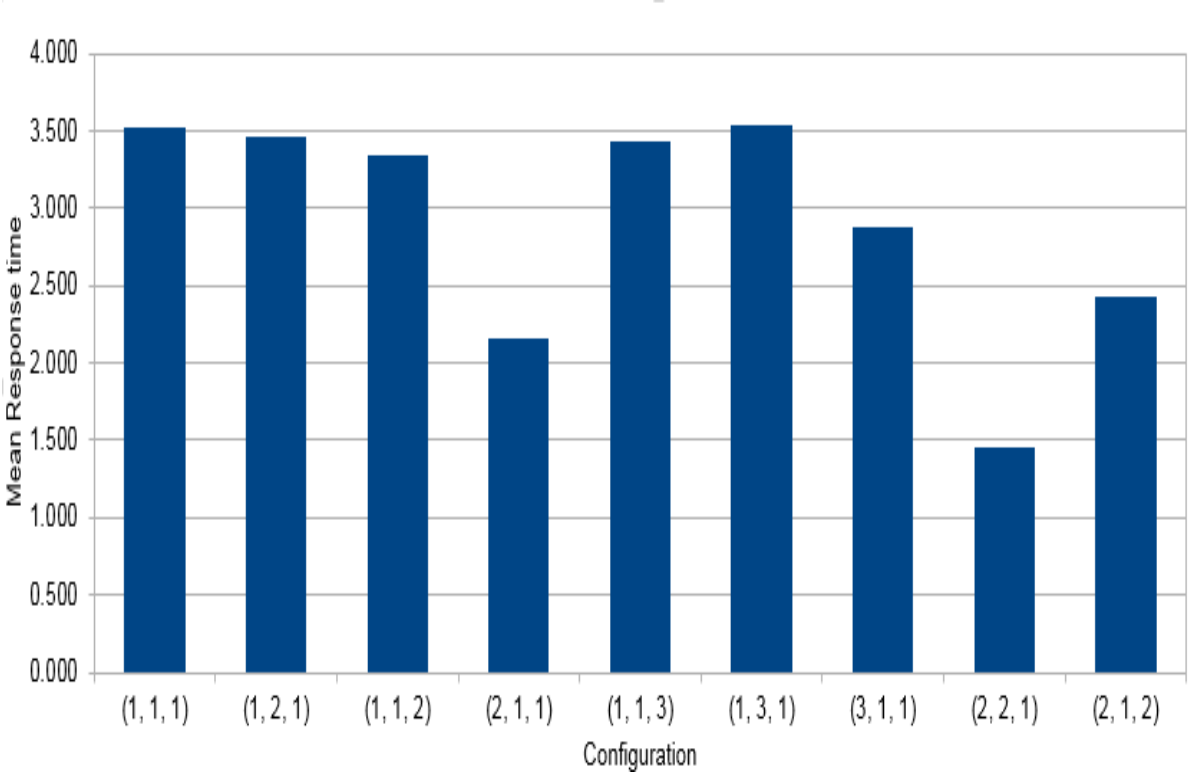

Fig 11: Average response times for 1st batch of VM configurations

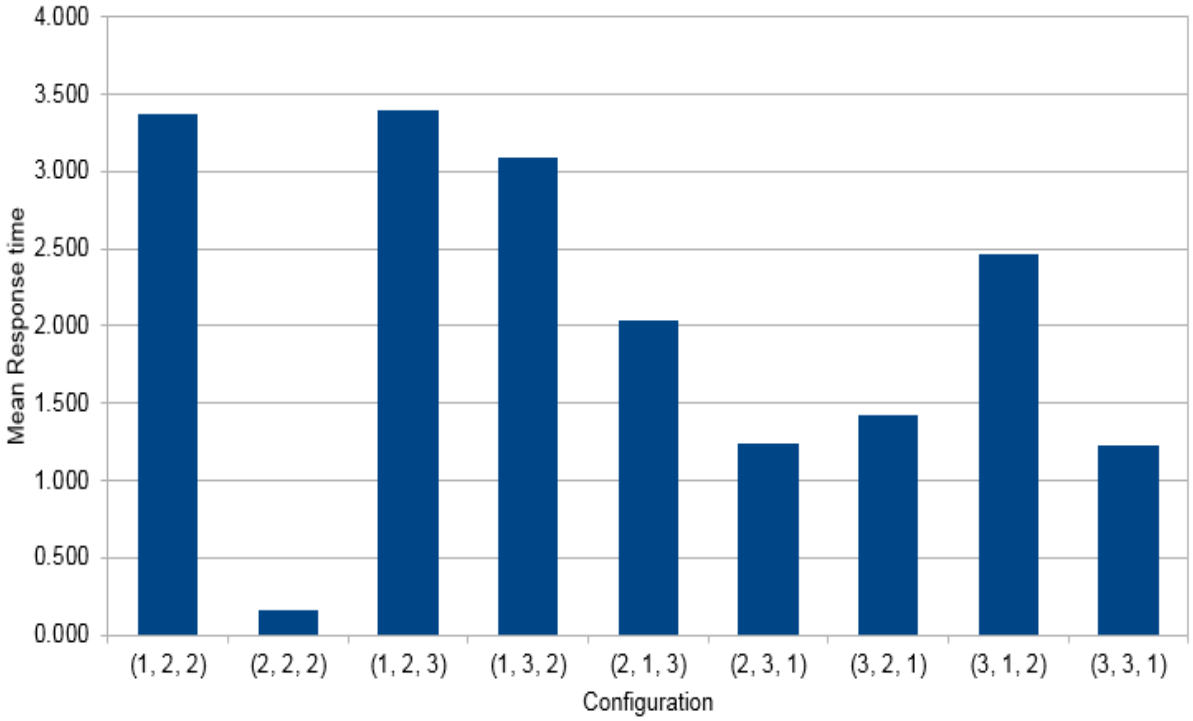

Fig 12: Average response times for 2nd batch of VM configurations

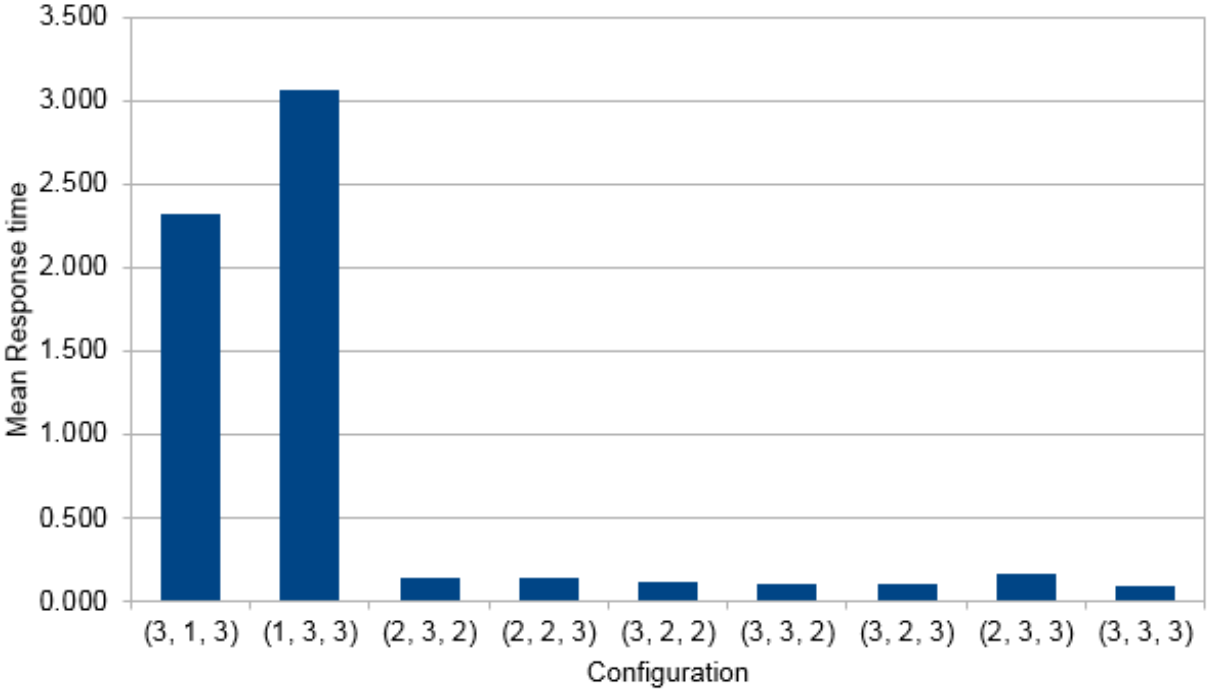

Fig 13: Average response times for 3rd batch of VM configurations

Figures 11, 12 and 13 are a graphical representation of table 1. Also, note that only 8 configurations from table 2 have a percentile of greater than 70% for a response time requirement range of 0.2 to 0.7 seconds. This is given in table 3. A graph plotting the percentiles for the 8 VM configurations, (2, 2, 2), (2, 3, 2), (2, 2, 3), (3, 2, 2), (3, 3, 2), (3, 2, 3), (2, 3, 3) and (3, 3, 3) in the three tiers as in table 3 is shown in the figure 14.

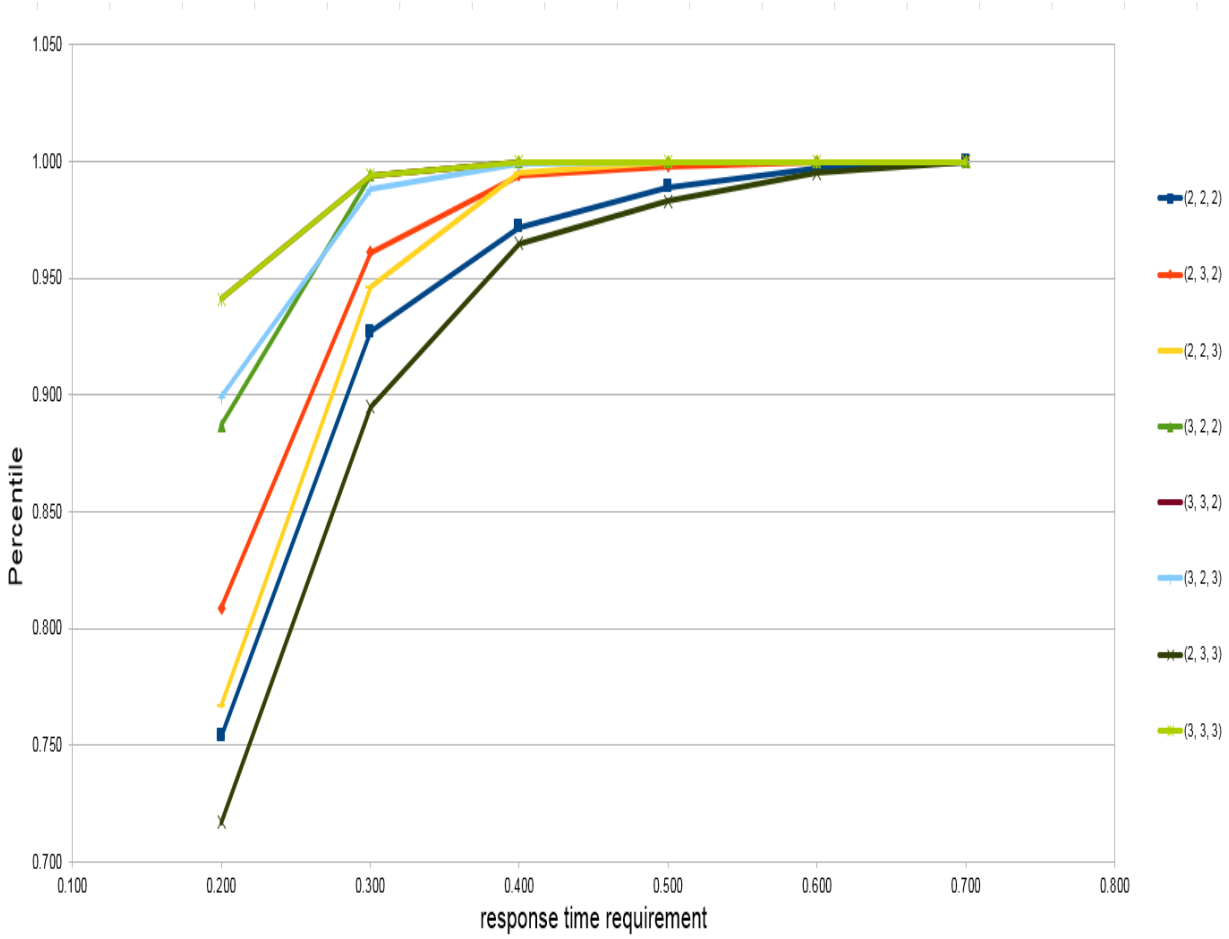

Fig 14: Graph of VM configurations' percentiles

The figure 14 shows that each of the 8 VM configurations had their percentiles gradually increase as the response time requirement increased from 0.2 to 0.7 seconds. Fig. 14 also shows that with a response time requirement of 0.7 seconds, all of the 8 considered VM configurations had a percentiles of 100%. However for VM configurations with response time

requirements between 0.2 to 0.6 seconds, as the number of VMs in the first tier increases, the percentiles approach close to 100%. Note that from table 3, the percentiles for the VM configurations, (3, 3, 2) and (3, 3, 3) are identical for the response time requirement range (0.2 to 0.7 seconds). Hence in the fig. 14, these two configurations overlap.

As an Application Service Provider (ASP), if a response time requirement of 0.3 seconds and 95% percentile was desired, then from fig 14, only 5 VM configurations, i.e. (2, 3, 2), (3, 2, 2), (3, 2, 3), (3, 3, 2) and (3, 3, 3) satisfy the requirement. Note that from table 3, the VM configurations ((3, 3, 2) and (3, 3, 3)) have overlapping values. The remaining 3 configurations, (2, 2, 2), (2, 2, 3) and (2, 3, 3) do not satisfy the above performance criteria.

Although their mean response times for all 8 configurations in fig. 14 are very low (i.e. lower than 0.2 seconds from table 1), it tells us that if an ASP wants the percentile to be greater than 95%, then the last 3 VM configurations ((2, 2, 2), (2, 2, 3) and (2, 3, 3)) will not satisfy the requirement. If the ASP wants more than 95% of incoming jobs to be processed for a mean response time lower than 0.3 seconds, then the (3, 3, 3), (3, 3, 2) and (3, 2, 2) are the best possible configurations (99.4% jobs processed) followed by (3, 2, 3) and (2, 3, 2) configurations.

However, if the ASP wants more than 95% of incoming jobs to be processed for a mean response time lower than 0.2 seconds, then from table 3 and figure 14, none of the 8 VM configurations satisfy this criteria.

Now, if the service rate of tier 1 (μ_1) = 90 jobs/second, service rate of tier 2 (μ_2) = 80 jobs/second and service rate of tier 3 (μ_3) = 70 jobs/second, i.e. $\mu_1 > \mu_2 > \mu_3$, we obtain the figure 15 below.

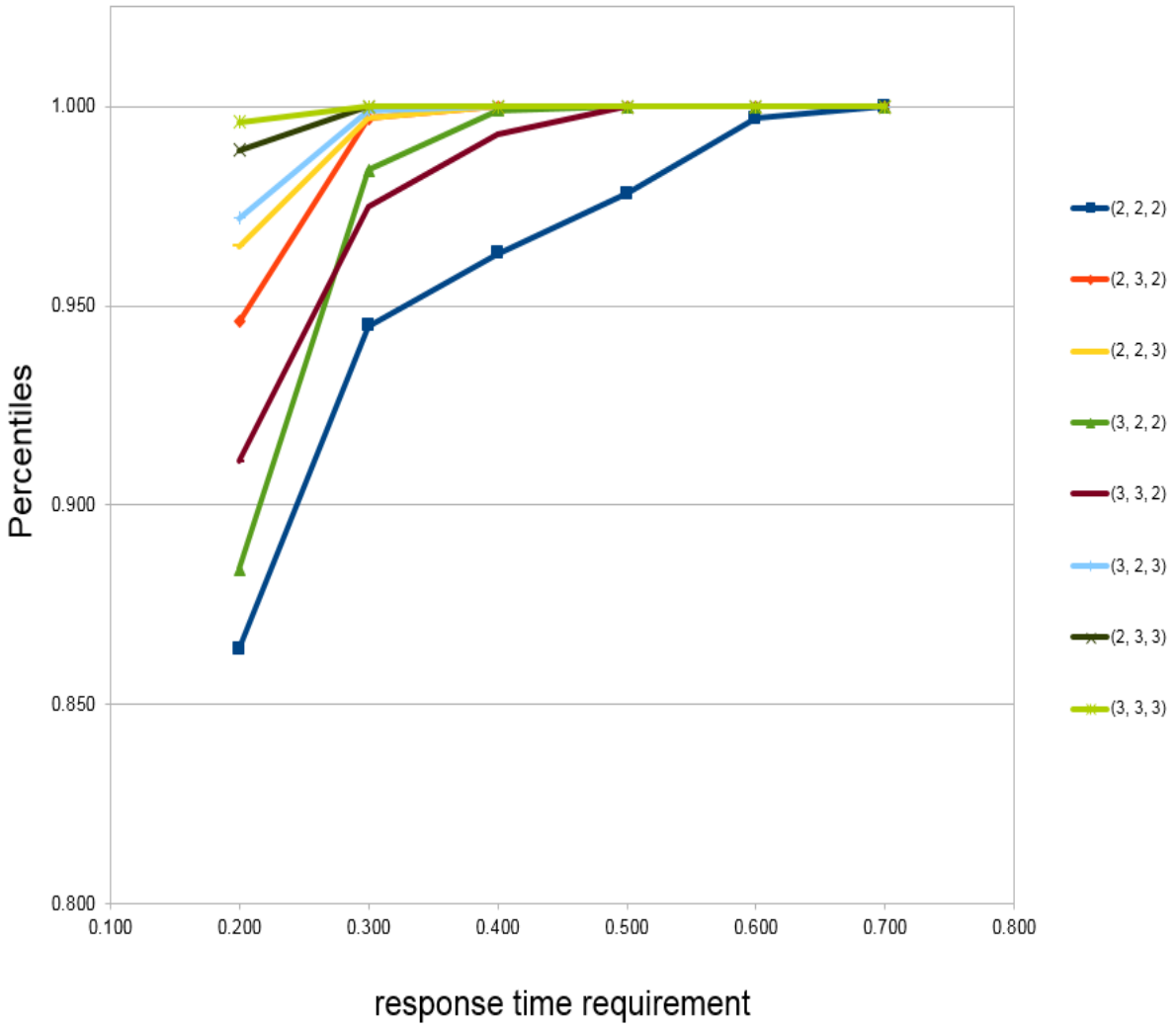

Fig 15: Graph of VM configurations' percentiles for (μ_1, μ_2, μ_3) = (90, 80, 70)

Note that from fig. 15, the VM configurations ((3, 3, 2) and (3, 3, 3)) don't have overlapping values. Also, from fig. 15, there are 4 configurations which satisfy the criteria of an ASP wanting more than 95% of incoming jobs to be processed for a mean response time lower than 0.2 seconds. These configurations are (3, 3, 3), (2, 3, 3), (3, 2, 3) and (2, 2, 3). Thus, the percentiles will be higher if $\mu_1 > \mu_2 > \mu_3$.

Thus, the mean response time performance metric doesn't give much information about overall system performance. Hence percentiles is very important as compared to mean response times.

CHAPTER 5

CONCLUSION

Discrete event simulation (DES) has the advantage of modelling the performance of a system using simulation rather than analytical methods. DES gives an approximate measure of performance of a system. The model in this paper allows application service providers to simulate their requirements and arrive at an ideal configuration or number of VMs they would need for their operations, which in turn they could request to be purchased from a cloud service provider. A 3-tier architecture was used in this research to model the architecture of an open queuing system. This queuing network was simulated using a discrete event simulation framework, simpy in order to find out how many replicas of VMs an ASP will need at each tier to handle a given workload of jobs. A threshold of between 0.2 to 0.7 seconds was used to determine which VM configuration had more jobs processed within that threshold. The results showed that for a large number of jobs to be processed, if the number of VMs were to be increased, it doesn't mean that the performance (viz. mean response time or percentiles) of the system is better. The performance of a system depends on the service rates, arrival rates as well as the number of VMs in any given tier. Within a set of VM configurations, if any one of the VM configurations had a comparatively lower number of VMs in any given tier, then that tier could act as a potential bottleneck for processing incoming jobs for that particular VM configuration. Also, for a set of VM configurations having the same total number of VMs in them, the VM configuration which has more VMs in the first tier was found to have a lower mean response time and higher percentile followed by the VM configuration that has more VMs in the second tier and so on. The results also showed that for a varying number of VMs

in a 3-tier open queuing network configuration, percentile was a more efficient performance metric as compared to mean response time. This was because details about the ratio of incoming jobs which were actually processed to the total number of jobs within a given response time range gives a customer or Application Service Provider more details than just mean response time, which could be low for all VM configurations.

Future work could consider performance impacts of failures of VMs at any tier (i.e. Tier 1, tier 2 or tier 3). Also, along with the model presented in this research, an approximation technique could be used in addition to an optimisation model to find an optimal configuration of a set of services rather than running the simulation repeatedly.

APPENDIX A

```
""" FCFS queues"""
from SimPy.Simulation import *
from math import *
from decimal import *
from random import expovariate, seed

## Model components ------------------------
class QN():
    def model(self, nrRuns, num, arrRate, endTime):
        averageRespTimeSum = 0.0
        probMetDeadlineSum = 0.0
        for runNr in range(nrRuns):
            self.wm1 = Monitor(name='ResponseTime')
            self.numberMeetDeadline = 0

            initialize()
            source = Source(self)
            activate(source,source.generate(num, arrRate, processors), at=0.0)
            simulate(until=endTime)

            if(self.wm1.count() > 0):
                result1 = self.wm1.count(), self.wm1.mean(), self.wm1.total()
                averageRespTimeSum = averageRespTimeSum + self.wm1.mean()
                print("Average response time for %3d completions was %3.3f
seconds. Total response time = %3.5f." % result1)
                probMetDeadline =
float(self.numberMeetDeadline)/self.wm1.count()
                probMetDeadlineSum = probMetDeadlineSum + probMetDeadline
                print("%d out of %d Jobs met the response time requirement of
%f" % (self.numberMeetDeadline, self.wm1.count(), responseTimeReq))
                print("Probability of Jobs that met the response time
requirement of %3.2f is %3.3f" % (responseTimeReq, probMetDeadline))
                print ("%s run(s) completed" %(runNr + 1))


print("**********************************************************************
***")
```

```
        print("Configuration is (%d,%d,%d)" %
(numberOfL1VMs,numberOfL2VMs,numberOfL3VMs))
        print("Average response time over %d runs is %3.3f seconds." % (nrRuns,
averageRespTimeSum/nrRuns))
        print("Probability of Jobs that met the response time requirement of
%3.2f over %d runs is %3.3f"
              % (responseTimeReq, nrRuns, float(probMetDeadlineSum)/nrRuns))

print("**********************************************************************
***")



class Source(Process):
    """ Source generates jobs randomly"""
    def __init__(self,sys):
        Process.__init__(self)
        self.sys = sys

    def generate(self, number, arrRate, processors):
        for i in range(number):
            c = Job("Job%02d" % (i), i, self.sys)
            activate(c, c.visit(processors))
            t = expovariate(arrRate)
            yield hold, self, t

class Job(Process):
    """ Job arrives, chooses the shortest queue
        is served and leaves
    """
    def __init__(self,name,i,sys):
        Process.__init__(self)
        self.name=name
        self.i=i
        self.sys = sys



    def visit(self, processors):
        arrivalToSystem = now()
        print ("%8.5f %s: Arrives     "%(now(),self.name))

        arrive = now()
        ##Job then moves to first level of servers
```

```
        p = numberOfL1VMs
        j = random.randint(1, p)
        yield request,self,processors[j-1]
        wait = float(now()-arrive)
        print ("%3.8f %s: Waited for %3.8f for %s"%(now(),self.name,wait,
processors[j-1].name))
        tiw = expovariate(serviceRateL1)
        yield hold,self,tiw
        yield release,self,processors[j-1]


        arrive = now()
        ##Job then moves to second level of servers
        q = numberOfL2VMs
        k = random.randint(1, q)
        yield request,self,processors[numberOfL1VMs+k-1]
        wait = float(now()-arrive)
        print ("%3.8f %s: Waited for %3.8f for %s"%(now(),self.name,wait,
processors[numberOfL1VMs+k-1].name))
        tiw = expovariate(serviceRateL2)
        yield hold,self,tiw
        yield release,self,processors[numberOfL1VMs+k-1]


        arrive = now()
        ##Job then moves to third level of servers
        r = numberOfL3VMs
        l = random.randint(1, r)
        yield request,self,processors[numberOfL1VMs+numberOfL2VMs+l-1]
        wait = float(now()-arrive)
        print ("%3.8f %s: Waited for %3.8f for %s"%(now(),self.name,wait,
processors[numberOfL1VMs+numberOfL2VMs+l-1].name))
        tiw = expovariate(serviceRateL3)
        yield hold,self,tiw
        yield release,self,processors[numberOfL1VMs+numberOfL2VMs+l-1]



        ## Job processing is done and the job leaves the network
        print ("%8.5f %s: Finished      "%(now(),self.name))
        responseTime = now() - arrivalToSystem
        print("%8.5f %s Response time: %2f" % (now(), self.name, responseTime))
        if responseTime <= responseTimeReq:
            self.sys.numberMeetDeadline = self.sys.numberMeetDeadline + 1
```

```
        self.sys.wm1.observe(responseTime)


## Parameters -------------------------

maxNumber = 1000
endTime = 2000.0    # seconds
serviceRateL1 = 60  # 60 jobs per second
serviceRateL2 = 70  # 70 jobs per second
serviceRateL3 = 80  # 80 jobs per second
arrRate = 100     # 100 arrivals per second
nrRuns = 3       # number of simulation runs
theseed = 787878
processors = []

responseTimeReq = 0.7
numberOfL1VMs = 3
numberOfL2VMs = 3
numberOfL3VMs = 3

for x in range(numberOfL1VMs):
    processors.append(Resource(name="L1VM"+str(x+1)))
for y in range(numberOfL2VMs):
    processors.append(Resource(name="L2VM"+str(y+1)))
for z in range(numberOfL3VMs):
    processors.append(Resource(name="L3VM"+str(z+1)))


## Model ------------------------------

seed(theseed)
plt=QN()
plt.model(nrRuns, maxNumber, arrRate, endTime)
```

REFERENCES

[1] G. Bolch, S. Greiner, H. de Meer and K. S. Trivedi, "Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications ", Second Edition, John Wiley, New York, NY, 2006. ISBN number: 0471565253.

[2] K. S. Trivedi, "Probability and Statistics with Reliability, Queuing, and Computer Science Applications ", 2nd Edition, John Wiley and Sons, 2001.

[3] R. Paharsingh and O. Das, "An availability model of a virtual TMR system with applications in Cloud/Cluster computing," in High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium on, 2011, pg. 261-268.

[4] R. Paharsingh and O. Das, "Availability analysis in virtual systems, with applications in cloud computing," in 2nd International Workshop on Cloud Computing and Scientific Applications (CCSA 2012), Ottawa, Canada, 2012.

[5] “https://simpy.readthedocs.org/en/latest/”, last accessed February 11th 2016.

[6] “http://heather.cs.ucdavis.edu/~matloff/simpy.html”, last accessed February 11th 2016.

[7] “http://www.ibm.com/developerworks/library/l-simpy/”, last accessed February 11th 2016.

[8] G. Wagner, O. Nicolae, J. Werner, “Extending discrete event simulation by adding an activity concept for business process modeling and simulation”, Proceedings of the 2009 Winter Simulation Conference (WSC), Austin, Texas, pg. 2951 – 2962, 13-16 Dec 2009.

[9] B. Pfitzinger, T. Baumann, D. Macos, T. Jestadt, “Using simulations to study the efficiency of update control protocols”, 2014 47th Hawaii International Conference on System Sciences (HICSS), Waikoloa, HI, pg. 5154 – 5161, 6 – 9 Jan 2014.

[10] E. Lazowska, “Quantitative System Performance, Computer System Analysis Using

Queuing Network Models” Prentice-Hall, Inc., 1984

[11] P. M. Broadwell, “Response Time as a Performability Metric for Online Services”, UC Berkeley Computer Science Technical Report, UCB CSD-04-1324, May 27, 2004.

[12] M. Grottke, V. Apte, K. S. Trivedi, S.Woolet, “Response Time Distributions in Networks of Queues”, International Series in Operations Research & Management Science, Book Chapter, pg. 587 – 641, 15th November 2010.

[13] B. Addis, D. Ardagna, B. Panicucci, M. S. Squillante, Li Zhang, “A Heirarchical Approach for the Resource Management of Very Large Cloud Platforms”, IEEE Transactions on Dependable and Secure Computing (Vol. 10, issue 5), pg. 253 – 272, 10th January 2013.

[14] A. N. Gullhav, B. Nygreen, P. E. Heegaard, “Approximating the Response Time Distribution of Fault-tolerant Multi-tier Cloud Services”, 2013 IEEE/ ACM 6th International Conference on Utility and Cloud Computing (UCC), Dresden, Germany, pg. 287 – 291, 9 – 12 Dec. 2013.

[15] O. Boxma, H. Daduna, “Sojourn Times in Queueing Networs”, Stochastic Analysis of Computer and Communication Systems, pg. 401 – 450, 1990.

[16] S. P. Woolet, “Performance Analysis of Computer Networks”, Ph. D. Dissertation, Duke University, Durham, NC, USA, 1993.

[17] G. Franks, T. Al-Omari, M. Woodside, O. Das, S. Derisavi, "Enhanced Modeling and Solution of Layered Queueing Networks", IEEE Transactions on Software Engineering (Vol. 35, issue 2), pg. 148 – 161, 5 Sep 2008.

[18] L. Carnevali, L. Ridi, E. Vicario, “A Quantitative Approach to Input Generation in Real-Time Testing of Stochastic Systems”, IEEE Transactions on Software Engineering (Vol. 39, issue 3), pg. 292 – 304, 26 Jun. 2012.

[19] R.Buyya, S. Garg, R. Calheiros, “SLA-Oriented Resource Provisioning for Cloud Computing – Challenges, Architecture and Solutions”, 2011 International Conference on Cloud and Service Computing, 12 – 14 Dec. 2011, Hong Kong.

[20] J. Nie, “A Study on the Application Cost of Server Virtualisation”, 2013 9th International Conference on Computational Intelligence and Security (CIS), 14 – 15 Dec. 2013, Leshan.

[21] L. Abeni, T. Cucinotta, “Efficient Virtualisation of Real-Time Activities”, 2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA), 12 – 14 Dec. 2011, Irvine, CA.

[22] M. Pretorius, M. Ghassemian, C. Ierotheou, “An Investigation into Energy Efficiency of Data Center Virtualisation”, 2010 International Conference on P2P, Parallel, Grid and Internet Computing (3PGCIC), 4 – 6 Nov. 2010, Fukuoka.

[23] S. I. Ahson, “Petri Net Models of Fuzzy Neural Networks”, IEEE Transactions on Systems, Man and Cybernetics (Vol. 25, issue 6), pg. 926 – 932, June 1995.

[24] L. Carnevali, E. Vicario, “A Quantitative Approach to Input Generation in Real–Time Testing of Stochastic Systems”, IEEE Transactions on Software Engineering (Vol. 39, issue 3), pg. 292 – 304, 26th June 2012.

[25] J. Xu, M. J. Chung, “Predicting the Performance of Synchronous Discrete Event Simulation”, IEEE Transactions on Parallel and Distributed Systems (Vol. 15, issue 12), pg. 1130 – 1137, Dec. 2004.

[26] B. Sharda, S. J. Bury, “Best Practices for Effective Application of Discrete Event Simulation in the Process Industries”, Proceedings of the 2011 Winter Simulation Conference (WSC), pg. 2315 – 2324, 11 – 14 Dec. 2011, Phoenix, AZ.

[27] S. H. Jacobson, E. Yucesan, “Common Issues in Discrete Optimisation and Discrete-Event Simulation”, IEEE Transactions on Automatic Control (Vol. 47, issue 2), pg. 341 – 345,

Feb. 2002.