Intrusion Detection and Prevention Systems

In the Cloud Environment

Muhammed Bello Abdulazeez

Department of Computer Science
University of Liverpool

This dissertation is submitted for the degree of
Doctor of Philosophy

August 2017

In the memory of Abdulazeez Umaru Dikko (Abba)

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgments.

The main part of this is based on four peer reviewed papers published or accepted for publication in different academic conferences.

1. Muhammed Abdulazeez, Pawel Garncarek, and Prudence W.H. Wong. Lightweight framework for reliable job scheduling in heterogeneous clouds. In *Proceedings of the 26th International Conference on Computer Communication and Networks (ICCCN-IoTPST)*, pages 1–6, 2017. [1]

2. Muhammed Abdulazeez, Dariusz R. Kowalski, Pawel Garncarek, and Prudence W.H. Wong. Lightweight robust framework for workload scheduling in clouds. In *Proceedings of the 1st International Conference on Edge Computing (EDGE)*, pages 1–4, 2017. [2]

3. Muhammed Bello Abdulazeez, Dariusz R Kowalski, Alexei Lisista, and Sultan S Alshamrani. Failure or denial of service? a rethink of the cloud recovery model. In *Proceedings of the 15th European Conference on Cyber Warfare and Security*, page 1, 2016. [3]

4. Muhammed Abdulazeez and Dariusz R Kowalski. Hierarchical model for intrusion detection systems in the cloud environment. In *Proceedings of the 14th European Conference on Cyber Warfare and Security*, page 319, 2015. [4]

Muhammed Bello Abdulazeez
August 2017

Acknowledgements

In the name of Allah the Beneficent the Merciful, I thank Allah for giving me the strength to conduct this research. Peace and blessing be upon his prophet Muhammad (SAW).

I have been able to complete this research program with the support and active co-operation of concerned bodies and several persons. I owe my debt and would like to express deep feelings of gratitude to my mentor and thesis supervisor, Professor Dariusz R Kowalski. I am intensely indebted to my second supervisor Dr Alexei Lisitsa. I will also like to thank Professor Prudence Wong for her assistance especially in the area of scheduling problems. My appreciation also goes to Paweł Garncarek of the University of Wrocław for some fruitful collaboration.

Besides my supervisors I will like to thank my advisers Professor Leszek Gasieniec and Dr Clare Dixon for providing valuable feedback throughout my studies. I will also like to thank my colleagues who supported me through this journey. In Particular Dr Sultan Alshamrani, Dr Mohamed Arikiez, Dr Austin Brockmeier, Dr David Hamilton, Pavan Sangha, Xenofon Evangelopoulos, Xia Cui, Yang Li, Peter Igwe and Yan Yan.

I have spent three and half years in Liverpool which was a long as well as hard time for my mother, Nafisa Abdulazeez, whose everyday prayers made it possible to complete this thesis. Finally I would like to thank my wife Maryam and my son Abdulazeez for putting up with me and my professional and academic commitments. Only she knows how much work has gone into this.

I gratefully acknowledge the financial support received from the Nigerian Government especially National Information Technology Development Agency (NITDA) for sponsoring my research and Nigerian Communications Satellite Limited my employers for supporting me during this research.

Abstract

Cloud computing provides users with computing resources on demand. Despite the recent boom in adoption of cloud services, security remains an important issue. The aim of this work is to study the structure of cloud systems and propose a new security architecture in protecting cloud against attacks. This work also investigates auto-scaling and how it affects cloud computing security. Finally, this thesis studies load balancing and scheduling in cloud computing particularly when some of the workload is faulty or malicious.

The first original contribution proposes a hierarchical model for intrusion detection in the cloud environment. Finite state machines (FSM) of the model were produced and verified then analyzed using probabilistic model checker. Results indicate that given certain conditions the proposed model will be in a state that efficiently utilize resources despite the presence of attack. In this part of work how cloud handles failure and its relationship to auto-scaling mechanisms within the cloud has been investigated.

The second original contribution proposes a lightweight robust scheduling algorithm for load balancing in the cloud. Here some of the traffic is not reliable. Formal analysis of the algorithm were conducted and results showed that given some arrival rates of both genuine and malicious traffic average queues will stabilize, i.e. they will not grow to infinity. Experimental results studied both queues and latency, and they showed that under the same conditions naive algorithms do not stabilize. The algorithm was then extended to decentralized settings where servers maintain separate queues. In this approach when a job arrives, a dispatching algorithm is used to decide which server to send it to. Different dispatching algorithms were proposed and experimental results indicate that the new algorithms perform better than some of the existing algorithms. The results were further extended to heterogeneous (servers with different configuration) settings and it was shown that some algorithms that were stable in homogeneous setting are not stable under this setting. Simulations monitoring queue sizes confirmed that some algorithms which are stable in homogeneous setting, are not stable under this setting.

It is hoped that this study with inform and enlighten cloud service providers about new ways to improve the security of the cloud in the presence of failure/attacks.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms / Abbreviations

API Application Programming Interface

AWS Amazon Web Services

CIA Confidentiality, Integrity and Availability

CPU Central Processing Unit

CTB Cloud Trace Back

CTBM Cloud Trace Back Mark

DARPA Defense Advanced Research Projects Agency

DDoS Distributed Denial of Service

DoS Denial of Service

EC2 Elastic Compute Cloud

FSM Finite State Machine

GRAND Greedy Random

HTTP Hyper Text Transfer Protocol

i.i.d Independent and Identically Distributed

IaaS Infrastructure as a Service

IDPS Intrusion Detection and Prevention System

IP Internet Protocol

JSQ Joint Shortest Queue

JSw Joint Shortest Work

MAC Media Access Control Address

NN Neural Network

NNS Neighborhood Negative Selection

OS Operating System

OSI Open System Interconnection

P2Q Power of Two Choices - Queue

P2W Power of Two Choices - Work

PaaS Platform as a Service

PM Physical Machine

RR Round Robin

SaaS Software as a Service

SLA Service Level Agreement

SOAP Simple Object Access Protocol

UF Utilization Factor

UR Uniform Random

VM Integer Linear Programming

VM Virtual Machine

XML Extensible Markup Language

Chapter 1

Introduction

1.1 Overview

The work in this thesis falls within the area of cloud computing security. More specifically, the aim is to study the structure of cloud defense systems and how security affects performance of workload management of cloud systems.

Cloud computing is a large scale distributed computing paradigm that is mainly driven by economies of scale, where a pool of abstracted, virtualized, dynamically-scalable, managed computing resources (e.g. networks, servers, storage, applications, and services) are delivered on demand to external customers over the Internet [6]. Whereas Mell et. al. [7] defined cloud computing as a computing paradigm that provides convenient, on-demand network access to a shared pool of configurable computing resources, which can be rapidly provisioned and released with minimal management effort or service provider interactions.

From the above definitions it can be observed that cloud computing inherits several properties from traditional computing paradigms such as Grid Computing [6], and it is delivered over the Internet. Therefore the security threats that affect the internet and other computing paradigms also affect cloud computing. Cloud computing however has its unique challenges due to it distinctive properties, such as dynamicity, scalability, size and heterogeneity [8].

The organization of this introductory chapter is as follows: Research questions, aims and objectives of this thesis are provided in Section 1.2. Section 1.3 provides motivation behind the research conducted in this thesis. The main research contribution and evaluation process are summarized in Section 1.4, then the summary of work published as a result of this research is summarized in Section 1.5. Finally the organization of the entire thesis is presented in Section 1.6.

1.2 Research Questions, Aims and Objectives

This section discusses the aims and objectives of this thesis. It concludes by proposing the research questions that this thesis will attempt to answer.

1.2.1 Aims

- Propose a hierarchical model for detecting intrusions in the cloud.
- Propose a robust scheduling algorithms for load balancing in cloud, these algorithms should efficiently allocate cloud resources even if some requests are faulty or malicious. Both centralized and decentralized approaches will be considered. The research will initially consider homogeneous data centers, but because data centers can be heterogeneous, this work will extend to heterogeneous data centers.

1.2.2 Objectives

- Study cloud computing security issues.
- Study cloud computing architecture.
- Study scheduling and load balancing techniques in the cloud.
- Study auto scaling techniques and how they are applied by cloud computing providers
- Propose a hierarchical model for intrusion detection in the cloud.
- Propose a new approach to auto-scaling in the cloud
- Compare the proposed models with existing approaches
- Propose scheduling algorithm for load balancing in the presence of failures and attacks.
- Analyze the propose algorithm.
- Conduct experiments to compare with naive algorithms
- Extend the proposed model to decentralized approaches and conduct experiments to compare different approaches.
- Further extend to homogeneous cloud and conduct experiments to compare with homogeneous setting.

1.2.3 Research Questions

This thesis aims to answer the following two main research questions, they are:

- How can the structure of cloud defense systems improve security?
- How unreliable or malicious workload affects *stability and reliability* of cloud data center?

1.3 Motivations

A strong motivation for this work comes from a number of papers. The work started by reading seminal papers on cloud computing, such as Mell et. al. [7], work by Weiss [9] and Armbrust et al. [10] that provide the foundation of cloud computing. Liu et. al [11] introduced cloud computing reference architecture. Influential papers on security of cloud computing that shaped the direction of this research are [12, 8, 13, 14].

Mulguri et. al [15] proposed a stochastic model for load balancing in the cloud. The authors showed that given certain arrival rates stability of queues can be achieved, this paper lays strong foundations and gives directions in the area of workload management in the cloud. The paper Somani et. at [5] wrote a survey paper which suggested that there is a relationship between workload management and security (especially availability) of cloud computing systems.

1.4 Research Contribution and Evaluation Process

There are two main contributions of this research to the body of knowledge. The first contribution is the design and analysis of a hierarchical model for intrusion detection in the cloud. The evaluation process of the hierarchical model was done by creating finite state machines and the stable state probabilities were calculated using the probabilistic model checker Prism [16].

The second contribution is a scheduling algorithm for load balance in cloud, which is able to efficiently schedule jobs and maintain stable queues despite the presence of failures and attacks. Both heterogeneous and homogeneous data centers were considered. The scheduling algorithm was analyzed theoretically and proofs where provided. Experiments were also conducted to show stability of the algorithm compared with [15], algorithm with naive security tool and algorithms without any security. Decentralized algorithms based on the main scheduling algorithm were also compared where some showed trends of stability while others did not.

1.5 Publications

This research has resulted in publication of the following papers:

- Muhammed Abdulazeez, Pawel Garncarek, and Prudence W.H. Wong. Lightweight framework for reliable job scheduling in heterogeneous clouds. In *Proceedings of the 26th International Conference on Computer Communication and Networks (ICCCN-IoTPST)*, pages 1–6, 2017. [1]

  The work in this paper is presented in **Chapter 7** of the thesis.

- Muhammed Abdulazeez, Dariusz R. Kowalski, Pawel Garncarek, and Prudence W.H. Wong. Lightweight robust framework for workload scheduling in clouds. In *Proceedings of the 1st International Conference on Edge Computing (EDGE)*, pages 1–4, 2017. [2]

  The work in this paper is presented in **Chapters 5 and 6** of the thesis.

- Muhammed Bello Abdulazeez, Dariusz R Kowalski, Alexei Lisista, and Sultan S Alshamrani. Failure or denial of service? a rethink of the cloud recovery model. In *Proceedings of the 15th European Conference on Cyber Warfare and Security*, page 1, 2016. [3]

  The work in this paper is presented in **Chapter 4** of the thesis.

- Muhammed Abdulazeez and Dariusz R Kowalski. Hierarchical model for intrusion detection systems in the cloud environment. In *Proceedings of the 14th European Conference on Cyber Warfare and Security*, page 319, 2015. [4]

  The work in this paper is presented in **Chapter 3** of the thesis.

1.6 Thesis Organization

The overall organization of the thesis is presented in this section. The first chapter is the introductory chapter and presents the overview of the entire thesis.

Chapter 2 provides necessary background work to this research. It also presents a review of the works that are related to this research. Some of the topics covered in this chapter include: introduction to cloud computing, security issues of cloud computing, and load balancing and scheduling in cloud computing data centers.

Chapter 3 presents a hierarchical model for intrusion detection in the cloud. This chapter is the foundation of the thesis and proposes a novel approach in handling security issues

of cloud computing environment. The proposed approach is presented as automata and the steady state probabilities of the automata are computed to ensure that it will be in certain 'desired' state given some attack scenario.

Chapter 4 analyses the relationship between load and security in cloud computing data centers. Popular cloud computing service providers has been surveyed and analyses of their auto-scaling features was done. A proposed approach of handling auto-scaling in the presence of failure will be proposed and evaluated.

Chapter 5 proposes centralized approaches for scheduling jobs in cloud computing data centers. A novel algorithm has been proposed that will consider some part of workload that is malicious (or unreliable). Some scanning strategy (to detect malicious or unreliable packets carrying job specification) will be proposed and it will be shown that given some input the proposed strategy will produce stable queues while naive scanning strategy will not. Both theoretical analyses and simulations will be conducted.

Chapter 6 extends the centralized approaches to decentralized approaches. Six different decentralized implementations are evaluated using the proposed scanning strategies in Chapter 5. It will be shown that some of the popular used decentralized implementations are not stable while some of the proposed ones are. Non-preemption will be introduced, i.e., when jobs start being processed they cannot be interrupted until they finish processing. Comparison of the centralized and decentralized approaches will be done and comparison of the preemptive and non-preemptive approaches will be conducted.

Chapter 7 presents decentralized algorithms for scheduling in heterogeneous cloud computing data centers. In this chapter has been be shown that some algorithms that can be stable for homogeneous data centers are unstable for heterogeneous data centers. Some theoretical analyses has been provided, supported by simulations.

Chapter 8 provides the summary of the entire thesis, highlighting the main contribution of the work in this thesis and proposing perspective areas for future.

Chapter 2

Security Issues of Cloud Computing

2.1 Overview

This chapter presents a review of the background work relevant to the results presented in this thesis. The aim of this chapter is to critically examine current security challenges relating to cloud computing and later to propose approaches to address some to the identified issues. The areas that will be addressed are: the architecture of cloud defense system, auto-scaling and how it affects cloud computing security, and finally job Virtual Machine (VM) scheduling and how it affects security of cloud computing.

The rest of the chapter is organized as follows: Section 2.2 provides an overview of cloud computing and, its security issues, while Section 2.3 describes some previously proposed solutions to the security issues. Job scheduling in cloud computing is discussed in Section 2.4. Finally, conclusion is presented in Section 2.5.

2.2 Introduction to Cloud Computing

2.2.1 Cloud Computing History

The history of Cloud Computing can be traced back to 1950s when users are able to access central computers using dumb terminals, this technology is called *Main Frame Computing* [17], this makes economic sense because it provide shared access to resources. Then came the virtualization technologies in the 1970s [18], these technologies make it possible to execute several distinct computing environments within the same physical machine, thereby taking mainframe computing to the next level. The next stage (in the 1990s) is the advent of virutalized private networks where telecommunications companies provide users with shared access to the same physical network infrastructure [19]. The notion of computing as a utility

was envisioned by Computer Scientist John McCarthy in 1961 [20], cloud computing is seen as realization of his dream. From the history above we can see that the cloud computing has been in existence for a while but the word cloud computing is said to be first used by George Favaloro and Sean O'Sullivan of Compaq [21].

2.2.2 Cloud Computing Definition

Cloud computing provides convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services), which can be rapidly provisioned and released with minimal management effort or service provider interactions [7]. This enables companies to increase capacity or add capabilities dynamically without investing in new infrastructure, training new personnel, or licensing new software [22]. From the definition above it can be seen that the cloud has the following characteristics:

- **On-demand self-service:** Resources are available when users request without the need for human intervention.
- **Broad network access:** Access to computing resources through different devices such as mobile phone, tablet, personal computers and any other devices with network access.
- **Resource pooling:** Resource assignment should not be fixed. There should be groups of resources for several different users that can be used based on need (load of applications) and if users do not need the resources they should release them for others.
- **Rapid elasticity:** The amount of resources should be able to be scaled up or down quickly based on the load of application.
- **Measured service:** The use of the service should be measured for payment and monitoring purposes.
- **Programmatic Access:** Users should be able to access services using trusted Application Programming Interfaces (API).

Cloud computing is an evolutionary outgrowth of existing computing approaches, which build upon existing and new technologies. The three main delivery models are *software as a service (SaaS)*, *infrastructure as a service (IaaS)*, and *platform as a service (PaaS)* [9, 23, 24]. According to [25, 26] there are three deployment models of cloud computing: *Private, Public and Hybrid Cloud*, while it is being argued there are in fact four, adding *Community*

Cloud [27, 11]. The four deployment models for cloud computing are briefly explained below:

- **Private.** This is where the cloud infrastructure is solely within a single organization, whether it is physically located within the organization or off premise.
- **Community.** Several organizations jointly construct and share the same cloud infrastructure as well as policies, requirements, values, and concerns.
- **Public.** This is the most common deployment model, where the cloud is used by the public and solely owned by a third party organization. Several popular cloud services are public e.g. Amazon EC2, Microsoft Azure, Google AppEngine and Rackspace.
- **Hybrid.** Here the cloud infrastructure is the combination of two or more of the cloud models above.

2.2.3 Cloud Computing Security

The cloud computing architecture combines three layers of interdependent infrastructure, platform and application [28]. Therefore, cloud computing and its services are becoming more attractive target to potential intruders due to its distributed and open nature [8]. Several studies have listed security as one of the key adoption issues of cloud computing [29–33, 22]. This is proven by several attacks targeted at cloud computing services. According to [34] cyber-criminals are moving to cloud. Top cloud computing service providers such as Sony [35], Microsoft [36], Rackspace [37] and Amazon [38] were all targets of attacks. Details of the features to these systems will be discussed in Chapter 4. Attacks targeted at Internet service providers affect users' ability to access cloud services, for example cyber attack that targeted at an Internet traffic company Dyn in October 2016 also affected Amazon [39]. Other criminals use cloud computing resources to lunch their attacks [40].

2.2.4 Denial of Service Attacks and Cloud Computing

The *Confidentiality, Integrity and Availability (CIA)* has been serving as the primary conceptual model for information security [41]. Saltzer et al. [42] defined *confidentiality* as unauthorized release of information, *integrity* as unauthorized modification of information and *availability* as unauthorized denial of access to information. Notice that all the attacks in Section 2.2.3 are aimed at unauthorized denial of access. This threatens one of the key advantages of cloud computing by preventing legitimate access to services normally provided by victims [43]. These types of attacks are called *Denial of Service attacks (DoS)*. A

Distributed Denial of Service attack (DDoS) attack deploys multiple attacking entities to prevent legitimate users from having access to system [44]. Figure 2.1 shows a scenario of DDoS in the cloud. *Application layer DDoS* are derived from low layers, these attacks utilize application layer protocols to overwhelm victim's resources [45], for this reason they are more undetectable [46]. With these threats to security of cloud computing there is a need for tools to assist prevent or at least detect these attacks.

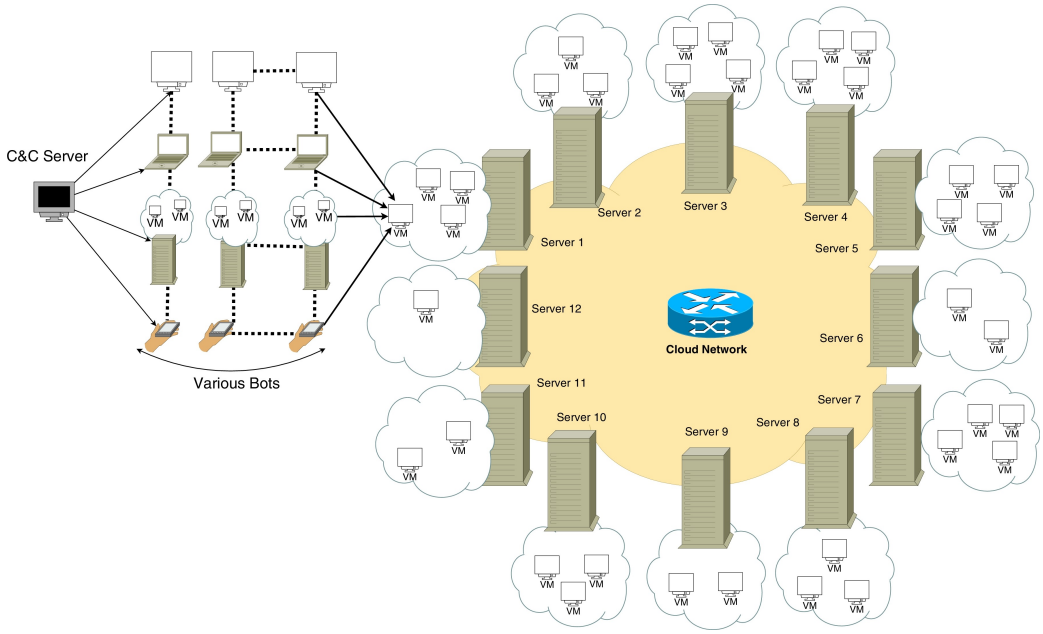


Figure 2.1 DDos attack scenario in cloud [5]

According to [5] *auto-scaling*, *pay-as-you-go accounting* and *multi-tenancy* are the three major reasons for adoption of cloud computing services. The authors argued that these three features are the reasons why attackers are successful in conducting DDoS attacks in clouds. The feature that is of interest to this research is auto-scaling, it will be discussed in more detail in subsequent chapters.

2.3 Intrusion Detection and Prevention Systems in Cloud

Intrusion Detection and Prevention System (IDPS) a tool that is used to improve security in the cloud environment. *Intrusion detection system* is a system that aims to detect attacks against computer systems and networks, or against systems in general, as it is difficult to provide secure information and maintain them in such a state for the entire lifetime and for every utilization [47, 48]. It becomes a prevention system when it not only detects attacks but it takes some actions to stop the attacks [49] as well.

To detect attacks two approaches can be used i.e., *signature/knowledge* or *anomaly/behavior* based detection [50]. In *signature* based detection, the detection system contains information about malicious/unreliable packets, when such packet is detected, alarm is raised or some action is being taken. This type of detection mechanism generally have high detection rate but tend to be slow because system needs to perform careful analyses of all possible malicious behaviors [47, 51]. *Anomaly* based detection attempts to model normal behavior. Any events which violate this model are considered to be suspicious [52].

On a quest to explore the cloud computing paradigm, there is a lot of research going on around cloud computing security. Chonka et al. [13] proposed a cloud security defense mechanism to protect cloud computing against HTTP-DoS and XML-DoS attacks. Placed at the edge routers in order to be close to the cloud network, the Cloud Trace Back (CTB) tool is used to locate the source of HTTP and XML DoS attacks and thereby reducing the risk of that particular node attacking the same system again. It is worthy to note that the CTB does not deal directly with the attacks, nor it prevents the attack messages from causing problems. This is done with the assistance of the Cloud Protector. The cloud protector is a trained back propagation Neural Network (NN), which helps to detect and filter HTTP and XML DoS attacks. Therefore the CTB depends on the ability of the cloud protector to detect attacks or it has to wait until an attack is successful before it can trace back the message. Because all incoming requests were sent through the CTB, which is placed before the web server, all packets were marked with Cloud Trace Back Mark (CTBM) tag within the CTB header. The CTB (in [13]) performed well in identifying the source of the attack, but the drawback is that is has to wait for an attack to occur before it can react (a reactive instead of proactive model). This is because it relies on the cloud protector to identify an attack, the cloud protector had a success rate of 91% with the miss rate of 9%. It took a minimum of 10 milliseconds (ms) to discover an attack, while the worst result was around $135 - 140ms$ mark.

Vissers et al. [53] proposed DDoS Defense System for Web Services in Cloud Environment. This defense system is placed at the entrance of the network, the cloud system is designed in such a way that it only listens to requests from the defense system. It starts by analyzing the user application request: if the request is found to be malicious, it is simply rejected. Otherwise, the request is passed on to the request handler and it is processed routinely by the system. The results presented by the authors did not provide any quantitative measure effectiveness of the system in detecting attacks, only that the system was able to detect the attacks. However, they demonstrated clear effect of the defense system on the attack as when malicious packets were sent to the server without the defense system, the CPU usage spikes to over 90% in the server and remains at that level while when the defense

system is activated the CPU usage spikes is 40% but within 10 seconds the usage stabilizes down to under 20%.

Chonka et al. [54] also proposed another solution for Detecting and mitigating HX-DoS attacks against cloud web services. The paper uses ENDER, which was an upgrade from previous research converted to a cloud application It applies two decision theory methods to detect attack traffic and mark attack message, namely CLASSIE and ADMU. CLASSIE was built from Pre-mark decision of IDP. ADMU is making decisions about the likelihood of message that has not been previously marked. According to the paper, ENDER was able to achieve a 99% detection rate to 2%; false positive, this performance is, however achieved under 50*s* which is a long time in the cloud environment. Another assumption is that collected traffic is clean from cross traffic and other attack traffic other than HTTP and XML based DoS.

Finally, Wang et al. [55] proposed exploiting Artificial Immune System to detect unknown DoS Attacks in real time. All the three previously described approaches were tested on a small low scale system. Therefore, this paper was studied to learn how DoS are detected in large scale systems. Due to the fact that payload based approaches are effective in detecting known attacks, but are less effective in detecting unknown attacks and payload analysis is a computationally expensive task especially on high speed networks. They proposed a detection scheme based on Artificial Immune Systems. The adopted Neighborhood Negative Selection (NNS) as the detection algorithm to detect unknown DoS attacks, and identify attack flows of massive traffic. To detect malignant traffic NNS extracts feature vectors for normal traffic to build a set. Then this set is used to divide shape space. NNS divides [0, 1] n into some fully adjacent but mutually disjoint neighborhoods. The union of these neighborhoods is the shape space of NNS. After division, the feature vectors of the normal flows are mapped into shape space. The authors used DARPA 1998 offline data-sets to test the system. The data sets consist of seven weeks of training data and two weeks of testing data. The system was able to detect about 91% of the attack traffic with some false negatives. Because normal data were used to train the system all attacks that were detected are unknown, which is impressive.

2.4 Job Scheduling in Cloud Computing

The aim of efficient task scheduling mechanisms in the cloud is to enhance overall performance. This is done by meeting users' requirements and improving resource utilization [56]. A lot of work has been done to improve scheduling in the cloud environment. Some of the major works are discussed below.

A class of algorithms look at an arriving job then decide which server to send the job for processing. The most widely used is the one that serves the job that needs most amount of resources first. This is generally called Best-Fit. Wang et al. [57] proposed a probabilistic multi-tenant model for virtual machine mapping in cloud systems, where they used Max-load-first algorithm, the same as Best-Fit. Using simulations the authors compared it with Min-load-first (here job that needs least amount of resources is served first) and randomize allocation of jobs (VMs). Results compared mean completion times of jobs and the authors found that the Max-load-first algorithm have the least mean completion time which suggests that it is the best algorithm. Rezvani et al. [58] proposed a similar scheme that uses Integer linear Programming (ILP). The ILP based algorithm showed better results and improved the utilization by 35% compared to a greedy, First-Fit and Worst-Fit algorithms. However, almost 20% of the resources in high loads are still not used in the ILP algorithm. Although the decision time of the ILP algorithm is 2.5 times faster than that of other algorithms, the decision time of all of them is acceptable for small data centers but not acceptable for large data centers.

Other authors attempt to solve the problem by looking at the servers instead of the properties of incoming jobs, a classic example is the *power-of-d-choices* (where d is the number of servers) with 2 as the most commonly used number. Here two servers are sampled uniformly at random and arriving jobs are sent to the server with the shortest queue. For example [59] showed that when arrivals are routed to the least utilized of $(d \geq 2)$ randomly selected servers, the blocking probability decays exponentially or doubly exponentially. This is a substantial improvement over the compared uniform random policy. In addition, they also developed an explicit upper-bound for the stationary fluid limit. The analysis of the upper bound revealed significant insight into the asymptotic behavior of large systems with the power-of-d-choices $(d \geq 2)$ algorithm. Authors in [60] also used the power-of-two-choices scheme but considered heterogeneous servers. The aim of the work is to reduce the average blocking probability of jobs in the system and their results showed that in the limiting system the servers behave independently—a property termed as propagation of chaos. Numerical results suggest that the proposed scheme significantly reduces the average blocking probability of jobs as compared to static schemes that probabilistically route jobs to servers independently of their states. Authors in [61] proposed similar approach with identical results to [60] but here a job is accepted for processing only if there is a vacancy available at the server to which it is assigned. Otherwise, the job is discarded or blocked and servers are heterogeneous.

Several other works also considered a class of problems that aim to minimize the number of occupied Physical Machines (PMs) [62, 63]. The aim of these algorithms is to reduce the

energy consumption of a data center because when servers are not used they consume less energy [63]. In [63], Stolyar et al., showed that a version of Greedy-Random (GRAND) is essentially asymptotically optimal, as the customer arrival rates grow to infinity. They also studied several versions of the GRAND algorithms through simulations. The first variant is such that there is an infinite number of servers of each type. Each arriving customer is assigned to a server immediately upon arrival. The second variant is a system with finite size pools of servers of each type. Each arriving customer can be either immediately assigned to a server or immediately blocked. In [62] the same authors proved that their result has a stronger form of asymptotic Optimality than that of [63].

Authors in [64] used genetic algorithm and fuzzy theory, the goal of the work was to improve system performance not efficient scheduling. Their approach improves system performance in terms of execution cost by about 45% and total execution time by about 50%. Xu et al. [65] looked at a case where each server maintains different queues (decentralized approach). They analyzed the collaborations of scheduling policies used in different phases i.e. ordering how jobs are processed when they reach a server and dispatching, which server to send a job when it arrives at the system. Shortest Expected Delay Routing Policy (SEDR) and Shortest Queue Routing Policy (SQR) are utilization optimal dispatching policies to collaborate with in the average case and worst case scheduling respectively; the collaboration of Shortest Remaining Time First (SRTF) ordering and SEDR dispatching achieves optimal throughput in the average case, while the collaboration of Myopic MaxWeight ordering and SQR dispatching achieves optimal throughput in the worst case.

Studies in [15, 66, 67] combined both job properties and server configuration when making scheduling decisions. They used the well known algorithm called MaxWeight where decision on which server to send arriving jobs is done based on queue lengths of different job types and server configurations. MaxWeight has been studied extensively in [67–71]. Maguluri et al. [15] proposed a stochastic model for load balancing in cloud environment where they proved stability for both preemptive and non preemptive cases. They also studied a version of the decentralized approach using join-the-shortest-queue and power-of-two-choices with MaxWeight scheduling. They also studied pick and compare with MaxWeight, which is suited for heterogeneous servers. In the paper they showed as well that both their algorithms are throughput optimal. Experimental analyses of mean delays also proved stability, given certain arrival rates of jobs. They also showed through simulations that the non preemptive algorithm is more stable for larger frame sizes.

Another work by Maguluri et al. [66] studies jobs with unknown duration arriving in clouds. In this work, they considered non-preemptive decentralized approaches and showed that refresh times occur often enough. They used this to show that this algorithm is throughput

optimal by showing that the drift of a Lyapunov function is negative. The authors studied the Joint Shortest Queue (JSQ) routing and routing algorithms with MaxWeight scheduling algorithm. It was known that these algorithms are throughput optimal. In this paper, authors showed that these algorithms are queue length optimal in the heavy traffic limit. Ghaderi [72] used randomized approach in allocating VMs to servers as opposed to MaxWeight. The randomized approach eliminates the problem of finding all possible configurations and simulation results show that the delay performance is better than MaxWeight. It is however non preemptive, decentralized and some comparisons where done with MaxWeight having global refresh times.

A study [73] related to security was done with the aim of optimizing recovery time after failures. When the system has a failure and that increases the time for recovery, for example, some physical machines have to restart suddenly due to the failure, it leads to increasing processing time for the work-flow because some task nodes results are lost, their proposal produces a work-flow schedule with the best performance compared with the others regardless of the number of tasks. Particularly, it achieves more than 22% of speed-up against the content aware scheduling which is a better algorithm in the absence of failure.

All the above works have made good attempts to improve efficiency of scheduling jobs in the cloud environment. From the literature review it was observed that none of the work considered scheduling jobs in the presence of failure. Therefore, this work will consider failure and attacks when scheduling jobs in cloud. The scheduling approach that will be considered is the one which considers both the job and the state of the servers because from the literature its performance seems to be better.

2.5 Summary

This introductory chapter introduced the notion of cloud computing and it main characteristics. It also discussed deployment and delivery models of this new technology. It then discussed the security threats cloud computing technologies listing some attacks that were successfully conducted on cloud data centers. After that IDPS was introduced as a tool that helps detect and prevent such attacks and then a review of several works on intrusion detection and prevention in the cloud environment. Finally literature review of several works on job scheduling in the cloud environment was conducted. Three different approaches were discussed i.e. approaches that consider jobs, approaches that consider server states and approaches that combine both the states of servers and incoming jobs before making scheduling decisions.

Chapter 3

Hierarchical Model for Intrusion Detection in the Cloud

3.1 Overview

This chapter introduces a lightweight, hierarchical, highly dynamic intrusion detection system that is well suited for cloud computing environment. The system uses application layer detection mechanisms to detect intrusions at different levels of the cloud computing hierarchy. Some rules were identified to detect the possibility of attacks on the application server. The checking of the rules is not fixed at certain components in the cloud, the system decides where to check based on the current load and the attacks detected in the component and the children nodes of the architecture. Intrusion detection load will be distributed across the cloud, eliminating single point of failure. The solution will address the heterogeneity challenge because servers (VMs) running different applications can have different detection approaches. For example, if a VM is running a windows OS, only signatures related to Windows vulnerabilities will be checked, and only Linux related rules will be checked for VM running Linux OS.

3.2 Proposed Model

This section presents the model, it starts with the classification of the rules that will be checked, then proposing the initial position they will be checked relative to the cloud computing architecture. Then the description of the new system architecture is given. Description of the rules and events is provided.

3.2.1 Rules checked

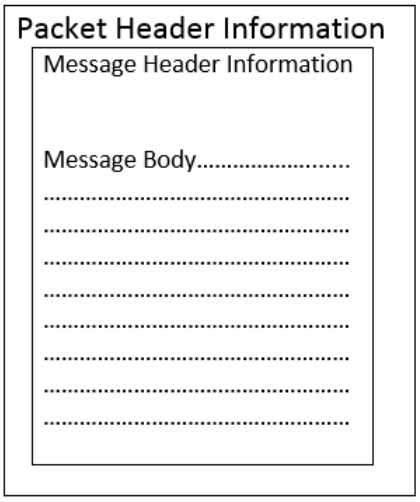


Figure 3.1 Typical packet structure

Packet can typically be separated into three broad areas i.e. packet header information, message header information and the message body. Figure 3.1 shows a typical packet separated into the three main components. Below is the description of the three components:

1. **Packet Header Information:** Network Level information of the packet such as sender's and receivers IP addresses, MAC addressed protocol and Time stamps in case of some specific protocols. Typically this information is at Network and Transport layers of the OSI Model.

2. **Message Header Information:** This provides information relevant to the application layer protocol used, in most cases it is a summary of the information in the packet. Typically this information is Application layer of the OSI Model.

3. **Message Body:** This is the actual message to be processed by the relevant application layer protocol. Typically this information is Application layer of the OSI Model.

   The widely used rule (signature) based intrusion detection system Snort has several rules that can be broadly classified as per the above classification [74].

3.2.2 Cloud Components and Rules

This section describes the rules and the location they will be checked within the cloud environment. Figure 3.2 shows which rules are checked at what position in the cloud environment.

Red box with solid circle indicates rules that must be checked at the corresponding position in the cloud, while green box with hollow circle indicates rules that can be checked anywhere across the architecture. For example, packet header information such as sender's IP address during predefined time t which is used to detect flooding, must be checked at the gateway of the cloud (Cloud Controller) while Message Content can be checked anywhere based on the security situation and current load of the cloud.

Cloud Components and Rules	Packet Information	Message Header Information	Message Content
Cloud Level	●	O	O
Cluster Level	O	O	O
Node (VM) Level	O	O	O

Figure 3.2 Rules and position that are checked in the cloud

3.2.3 New System Architecture

This section introduces the new cloud security architecture which has three components, namely: the defense system, monitoring systems and the state of the cloud architecture. The interaction among these subsystems is depicted in Figure 3.3 and described below:

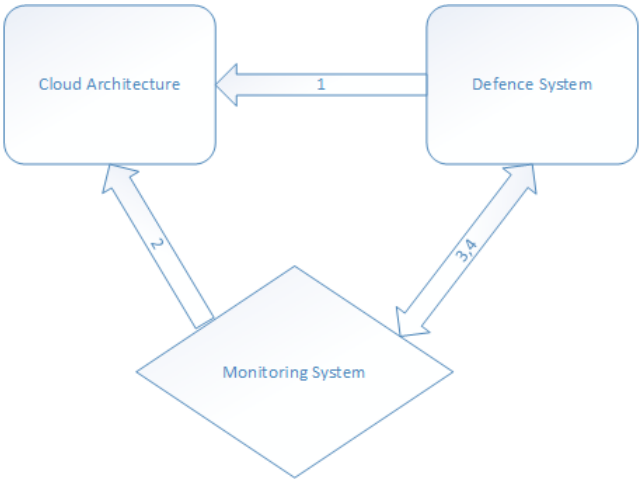


Figure 3.3 High level architecture of the proposed model

1. This is the Initial stage of the defense system where the assignment of checking the rules is done as follows:

   (a) At the gateways of the cloud the system it checks the simple rule of packet information, here flooding and blacklisted IP addresses may be checked. The cost of checking these rules is constant time. Several cloud defense systems have only this feature [13, 54].

   (b) At the cluster level, message header properties are checked, for example SOAP (Simple Object Access Protocol) properties include Soap Action and Content length and other summary information about the message, the cost of checking these rules is constant time also.

   (c) Finally, inspection of the whole message body is done at the nodes, this is because the nodes will have to process the message anyway and the cost of pre-processing is saved at each level of the cloud. The rules checked here usually require linear time (relative to the length of the message)to process.

2. The second component is where the monitoring system monitors the cloud architecture in terms of the load of each component and updates the defense system on where to check rules.

3. In this stage the defense system reports to the monitoring system the attacks it has discovered and their location. This is done concurrently with the second stage.

4. Finally the monitoring system updates the defense system based on the information it has received from 2 and 3 above. The policy in which the defense system changes the location of where to check which rules is discussed in the following section.

3.2.4 Events

This sections describes the series of events that can trigger change of state of one of the cloud component. Most of the events are based on the notion of attack been detected in one of the devices of the cloud. It is assumed that there are available IDS that can be configured to detect attacks [74]. The events are listed in the table below:

Table 3.1 Description of events that trigger actions in the automata

ID	Event
E1	Attack detected towards single Node and Node is in S1
E2	Attack detected towards multiple Node of the same Cluster

Table 3.1 Description of events that trigger actions in the automata

ID	Event
E3	Attack detected towards multiple Nodes of different Clusters
E4	Attack detected towards single Cluster
E5	Attack detected towards multiple Clusters of the same Gateway
E6	Attack detected towards multiple Clusters of different Gateways
E7	Attack detected towards Single Gateway
E8	Attack detected towards Multiple Gateways
E9	If cluster has more than x nodes running under it and no security event has occurred from last v seconds.
E10	If Gateway has more than y clusters running under it and no security event has occurred in the last w seconds.
E11	If node spends u seconds without any security event and Node is in S2
E12	If Cluster spends v seconds without any security event Cluster is in S2
E13	If Cluster spends v seconds without any security event Cluster is in S2
E14	Attack detected towards single node and node is in S2
E15	Attack detected towards single node and node is in S3
E16	If node spends u seconds without any security event and Node is in S3
E17	If node spends u seconds without any security event and Node is in S4
E18	If Cluster spends v seconds without any security event and Cluster is in S2
E19	If Gateway spends w seconds without any security event and Gateway is in S3
E20	If Gateway spends w seconds without any security event and Gateway is in S2

3.2.5 States

The states depict the level of criticality, the more critical a state is the more expensive the cost of doing security procedure is. For example when the gateway is in state S1 checks packet header information which is done in constant time if it is in S2 the it checks the message header information which is done also in constant time to the length of the message (slightly more expensive than packet information checks). When it is in S4 we check the entire message which is linear to the length of the entire message. This can be expensive because actions like deep packet analyses are conducted here [75]. The parameters v, u, w

Table 3.2 Different states of the cloud components

Severity	Alertness State	Alertness Severity
low	S1	Attack is unlikely (most efficient state)
substantial	S2	Attack is possible but not likely
severe	S3	Attack is a strong possibility
critical	S4	Attack is expected imminently (least efficient state)

can be set by administrator based on the security requirements of the application. This is because different applications have different security requirements. Those with more stringent security requirements will require less nodes (i.e. low value of x) to be under attack for the application to move to more critical state.

3.2.6 Automata

This section presents the events and states presented in form of FSM. Figures 3.4, 3.5 and 3.6 present the FSM for the three components of the of the cloud computing hierarchical, which Figures 3.3, 3.4 and 3.5 show the transition tables. The automata presents how the cloud computing components respond to the events. Changing to higher criticality levels in case of attacks and decreasing in criticality in the absence of attacks.

Table 3.3 Transition table for node

	S1(t +1)	S2(t +1)	S3(t +1)	S4(t +1)
S1(t)	o	E2	-	E1
S2(t)	E11	o	E3	E14
S3(t)	-	E16	o	E15
S4(t)	-	-	E17	o

Table 3.4 Transition table for cluster

	S1 (t +1)	S2(t +1)	S3(t +1)	S4(t +1)
S1(t)	o	E1	-	E4
S2(t)	E18	o	E2	-
S3(t)	-	E12	o	E3
S4(t)	-	-	E9	o

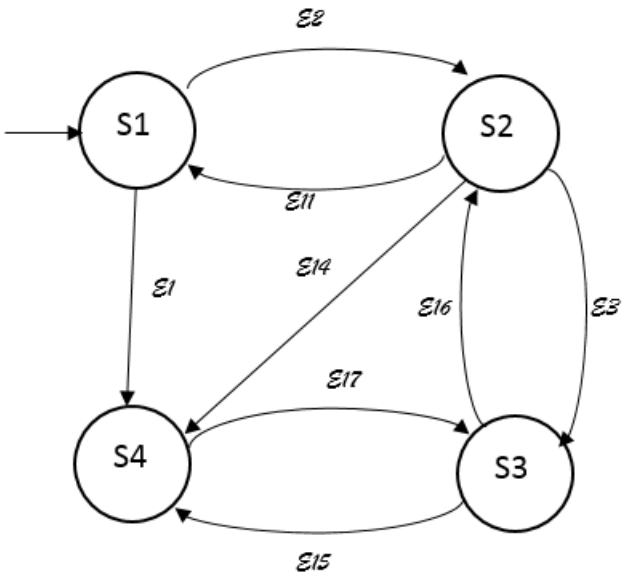


Figure 3.4 Finite state automata for the node

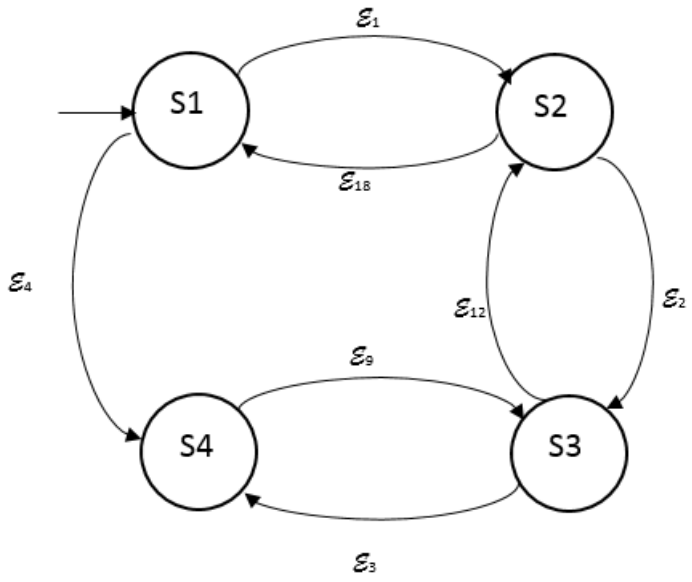


Figure 3.5 Finite state automata for the cluster

3.3 Model Checking

To verify the safe and reliability of the models proposed, the above models where constructed and verified using the probabilistic model checker Prism [16]. Given a probabilistic model of a system, Prism can be used to analyze both temporal and probabilistic properties of

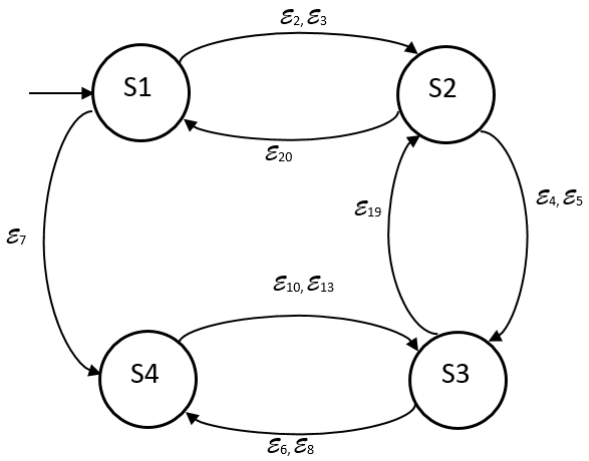


Figure 3.6 Finite state automata for the gateway

Table 3.5 Transition table for gateway

	S1(t +1)	**S2(t +1)**	**S3(t +1)**	**S4(t +1)**
S1(t)	o	E2, E3	-	E7,
S2(t)	E20	o	E4, E5	-
S3(t)	-	E19	o	E6, E8
S4(t)	-	-	E10, E13	o

the input model by exhaustively checking some logical requirement against all possible behaviors. Therefore, Prism was used to compute the steady state probabilities of each of the FSMs. The primary aim of the model checking is to verify that given a detection system the proposed model what is the probability of the system being in S1, that is in a state that consumes less resources. This however is dependent on how often attacks come into the system, therefore the probability of attack detected towards the node is varied and how the steady state probability changes is observed.

3.3.1 Experimental Setup

The description of the experiment setup is provided in this section. In the experiment for the node automata, the probability of event E1 happening is varied between 0.1 and 0.01. E1 is used because the other 'bad' events are all aimed attacking a single node. Experiments were ran for 100000 times.

Table 3.6 Steady state probabilities of node automaton

Probability of Attack	state 1	state 2	state 3	state 4
0.01	0.83848	0.11978	0.02611	0.01563
0.02	0.80974	0.12724	0.03708	0.02594
0.03	0.78290	0.13421	0.04732	0.03557
0.04	0.75778	0.14073	0.05690	0.04459
0.05	0.73422	0.14684	0.06589	0.05304
0.06	0.71209	0.15259	0.07434	0.06099
0.07	0.69124	0.15800	0.08229	0.06847
0.08	0.67159	0.16310	0.08979	0.07552
0.09	0.65302	0.16792	0.09688	0.08218
0.1	0.63545	0.17248	0.10358	0.08849

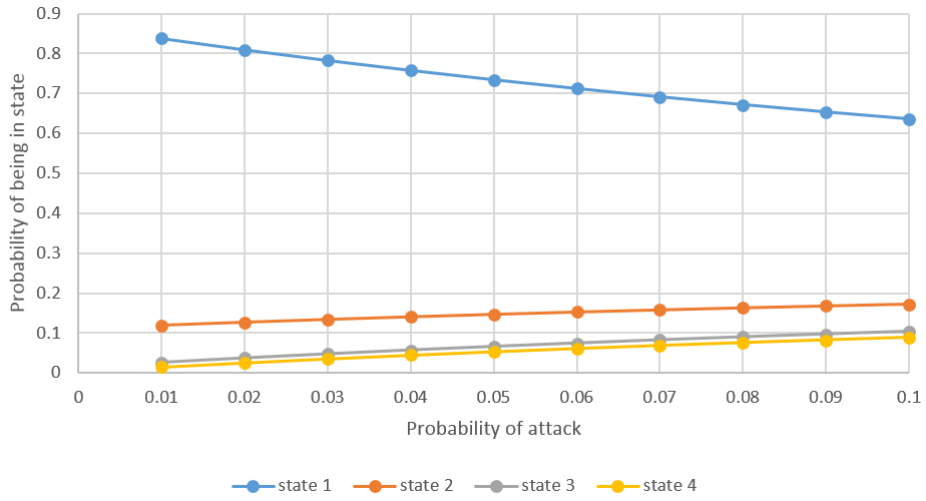


Figure 3.7 Graph showing the steady state probabilities for node automaton

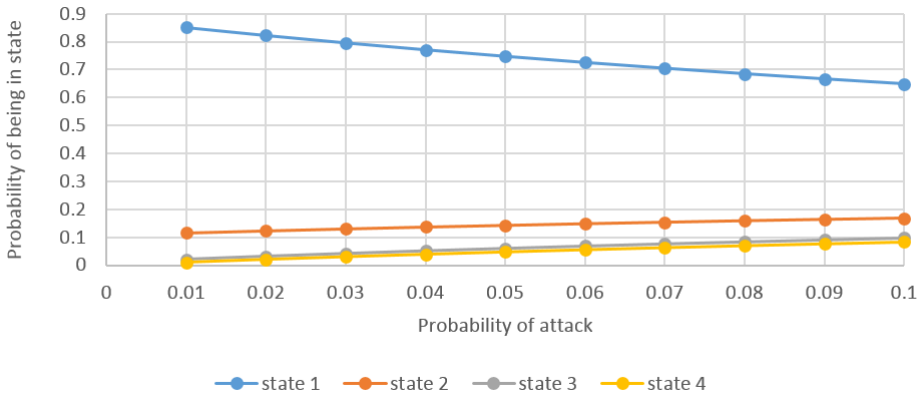


Figure 3.8 Graph showing the steady state probabilities for gateway automaton

Table 3.7 Steady state probabilities for gateway automaton

Probability of Attack	**state 1**	**state 2**	**state 3**	**state 4**
0.01	0.85090	0.11656	0.02137	0.01117
0.02	0.82248	0.12394	0.03221	0.02137
0.03	0.79591	0.13083	0.04235	0.03091
0.04	0.77099	0.13730	0.05186	0.03985
0.05	0.74759	0.14337	0.06079	0.04824
0.06	0.72557	0.14909	0.06919	0.05615
0.07	0.70481	0.15448	0.07712	0.06360
0.08	0.68520	0.15957	0.08460	0.07063
0.09	0.66666	0.16438	0.09167	0.07729
0.1	0.64909	0.16894	0.09838	0.08360

3.3.2 Results

Finite state automata for the three components of the cloud and their different states were presented in Section 3.2.6. This provides discussion of the automata and the results of the experiments. The automata is designed in such a way that when attack is detected at a device the reaction is to change the state of that device to the state with highest alertness. When attack is detected in the child node of a device it is put in an increase alert state but not the highest, if however several devices with the same parent are reporting attack the device in put into more alert level. For example if attack is detected at the a node the cluster that that node belongs to is put in an increase alert level, if more nodes of the same cluster are reporting attack the cluster is put in even more alert state.

From Figure 3.7, it can be observed that as the number of malicious packets increase the probability of the system being in undesirable state increases. The trend is similar in Figure 3.8 for gateway, this is expected because the same high-level idea that is used for the both automata.

3.4 Summary

In this chapter a hierarchical model for intrusion detection in the cloud was introduced. The model leverages existing cloud architecture to implement intrusion detection by checking different rules at different levels of the cloud hierarchy based on the current security situation and load of applications. The model was then presented as FSM and results indicate that the system will stay in a stable state even in the presence of attacks.

Chapter 4

Auto Scaling, Failure and Recovery in Cloud

4.1 Overview

This chapter explores how current cloud platforms handle failure. Status monitoring or accounting is an essential component for the smooth operation of today's cloud data centers, as quick responses to anomalies, failures or load have crucial business and performance ramifications.

The rest of this chapter is organized as follows: Section 4.2 discuses related work, application layer DoS and auto-scaling. Section 4.3 surveys auto-scaling features of major cloud service providers. Section 4.4 compares the auto-scaling features of the major cloud providers. Section 4.5, presents a new security architecture for cloud computing. Assessment of the new architecture and comparison with the existing architecture is done in Section 4.6. Finally, conclusions are discussed in Section 4.7.

4.2 Overview of Auto Scaling

The management of computing elasticity is usually an application specific task and involves mapping an application's requirements to the available resources. The process of adapting resources to on-demand requirements is called scaling [76]. Under-provisioning of resources hurts performance, which can lead to Service Level Agreement (SLA) violations while over-provisioning results in idle resources which leads to incurring unnecessary costs. A natural thought will be to provision for average load or peak load. Average load planning is cost effective but when peak load occurs performance is negatively affected which can lead

to disgruntled customers. Planning for peak load ensures performance never suffers but it is not cost efficient. Table 4.1 summaries the benefits and the drawbacks of using average or peak load when provisioning applications.

Table 4.1 Manual scaling techniques

Load	Pros	Cons
Average	Low cost	Poor performance during peak periods
Peak	No negative impact on performance	High cost

Thus, it is necessary to come up with a more sophisticated method for efficient and cost effective way to scale an application's resources according to demand. The act of dynamically scaling based on demand of applications is called *auto-scaling* [77]. Essentially, there are two approaches to auto-scaling.

4.2.1 Schedule Based Mechanisms

Here, the cyclic pattern of daily, weekly or monthly workload is taken into account and provisioning is done based on the workload [78]. The drawback of this method is that it cannot handle unexpected changes in loads. Many cloud providers offer scheduled based auto-scaling mechanisms, this is especially useful to customers who can reliably predict the load of their applications.

4.2.2 Rule Based Mechanisms

In this approach two rules are created to determine when to scale up or down. Each rule is user defined and the condition is based on target variable, for example if the CPU load is greater than 75% [76]. When this happens pre-defined action is triggered e.g., adding a new VM. This form of auto-scaling is classified as reactive because it waits for load to increase before it reacts. Other techniques are available that try to anticipate future needs which are called *predictive or proactive auto-scaling* [79]. Major cloud providers mainly use the reactive rule based auto-scaling techniques.

4.3 Analyses of Auto-Scaling and Failure Recovery Features of Cloud Providers

4.3.1 Amazon Web Services

Amazon provides its *IaaS* through its Elastic Compute Cloud (EC2) [80]. An important feature of rule-based auto-scaling in AWS is CloudWatch. CloudWatch monitors the applications that run on AWS in real time and can be used to collect and track various metrics of the applications for users [81] CloudWatch only reports the metrics, no further information is given if there is high resource utilization.

Amazon achieves auto-scaling through different approaches:

- If users have predictable load auto-scaling can be achieved by scheduling scaling plans based on the known load changes. This uses the schedule based techniques.
- Another way the EC2 achieves auto-scaling is by checking when average utilization of the EC2 is high then more instances are added, similarly conditions can be set to remove instances when the utilization is low. No mechanism to check the cause of high utilization of resources. This is a form of reactive rule based auto-scaling.
- A more sophisticated way to achieve auto-scaling in EC2 is through the Elastic Load Balancing, this helps to distribute instances within auto-scaling groups [82]. It uses CloudWatch to send alarms to trigger scaling activities.

4.3.2 Microsoft Azure

Initially Azure was a PaaS provided by Microsoft, offering Webrole and WorkerRole for hosting front-end applications and processing of backend workloads. Recently it has allowed users to deploy a Windows image prepared offline in the cloud. In this approach called VMRole; users can control the entire software stack and are able to remotely access the VM. This makes the VMRole effectively an IaaS type of service. Moreover, unlike other PaaS cloud server providers, Microsoft has added APIs and enabled remote desktop connections to log into hosting operating systems, which functions more like a virtual machine [83]. The Azure platform provides manual scaling through the Azure management portal. Users can also scale application running all the types of Virtual Machines, i.e., WebRole, WorkerRole and VMRole. To scale an application running any of the instances above users can add or remove role instance to accommodate the work load. When applications are scaled up and

down, there is no creation or deletion of new instances instead a set created previously are turned off and on from an availability zone [83].

4.3.3 IBM Smart Cloud

IBM uses OpenStack to provide auto-scaling through a feature called Heat. This feature reduces the need to manually provision instance capacities in advance. Users can use Heat resources to detect when a Ceilometer alarm triggers and provision or de-provision a new VM depending on the trigger. These groups of VMs must be under a load balancer which distributes the load among the VMs on the scaling group. IBM Smart Cloud also allows users to manually scale applications based on the work load. Users are also able to scale applications based on predicted schedule [84].

4.3.4 Rackspace Open Cloud

Rackspace is another free and open source service. This is also done through OpenStack, initially with collaboration with NASA. However, several other companies have joined the OpenStack Project including IBM mentioned above. Rackspace also provides auto-scaling based on pre-defined rules on schedule. It does not provide auto-scaling based on dynamic load changes (i.e., it does not provide rule based auto-scaling). Another form of scaling that is provided by Rackspace is Webhook (capability-based URL), in this case scaling occurs when a URL is triggered [85]. This however, can only be classified as manual scaling. Rackspace monitors metrics using the Monitoring Agent [86]. Agents can be installed in the cloud servers that users want to monitor. The checks that are available for users include HTTP, TCP, ping, memory, CPU, load average, file system and network [87].

4.3.5 Google Compute Engine

Google compute engine uses managed instance groups to offer auto-scaling capabilities that allow users to automatically add or remove instances from a managed instance group based on increases or decreases in load. To create an auto-scaler, users must specify the auto-scaling policy the auto-scaler should use to determine when to scale. Users can choose to auto-scale using the following policies:

- Average CPU utilization
- Cloud monitoring metrics

- HTTP load balancing serving capacity, which can be based on either utilization or requests per second.

Users can also scale cloud the resources up or down manually or using schedules if they can predict future changes in load [83].

4.4 Comparison of the Auto-Scaling and Security Features

This section discusses the *auto-scaling* features provided by cloud providers with respect to security.

Firstly, all the providers allow users to manually scale their applications based on needs. The second common feature of all cloud providers is schedule based *auto-scaling*, where users can predict load increases/decreases at different times and scaling is done based on these changes. It is a form of proactive *auto-scaling*. In all but one (Rackspace) of the cloud vendors users can automatically scale their applications based on some metrics, i.e., true auto-scaling where the resources are scaled on the fly based on current application needs. This however, is reactive and it does not predict when load changes. This, can take a while because it takes between 44.2 to 810.2 seconds to start an instance of a virtual machine, depending on the cloud provider and the type of VM in question [88].

Another critical feature the providers are yet to address is the cause of change in load. Where there is an increase in CPU load utilization, HTTP requests per unit time or other metrics, alarms are simply raised by the monitoring system e.g. CloudWatch in EC2 or Heat in IBM and application capacity is increased. The monitoring system does not check for the reason of the increased in load it just alerts the auto-scaler to increase resources to accommodate the increase in load. This is a good feature if the event occurred due to increased business activity but it is not fair especially to the cloud customer if it occurs as a result of malicious activity. Figure 4.1 illustrates the current cloud architecture.

An interesting finding during this survey is in the documentation of EC2 Auto-scaling developer guide [89]. Amazon provides health checks for all its VMs. Virtual Machine in EC2 has two states, *Healthy* and *Unhealthy*. However, Amazon does not provide a formal definition for the states in their documentation. Moreover, there is no mechanism to detect what causes the *Unhealthy* state of the VM.

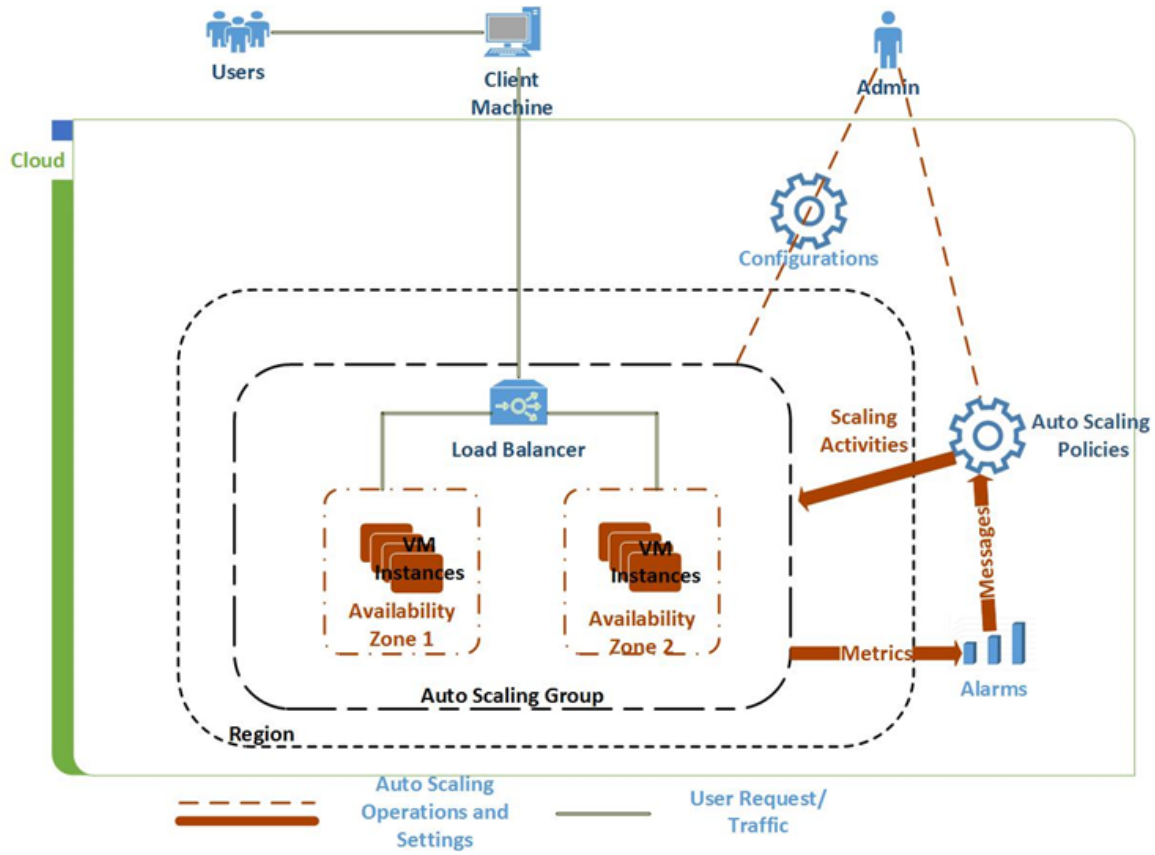


Figure 4.1 Current cloud architecture

4.5 Proposed New Architecture

The cloud vendors have made a lot of effort to monitor the performance of the cloud, to automate all processes of scaling VMs and to provide customers with uninterrupted service. What they have not done yet is to provide reason for failure of virtual machines. This can be unfair to cloud customers if the reason of a failure is as a result of application layer DoS and not true increase in load of application. When this occurs and resources are scaled up, customers can be overcharged and occurrences of similar events might not be prevented. This chapter proposes two different approaches to tackling this problem.

4.5.1 Use of Current Metrics

All the cloud vendors surveyed in this paper provide metrics on the use of resources in the cloud. These are important because the metrics can be analyzed in order to determine why a virtual machine failed or why it using a lot of resources. Using two metrics in this thesis:

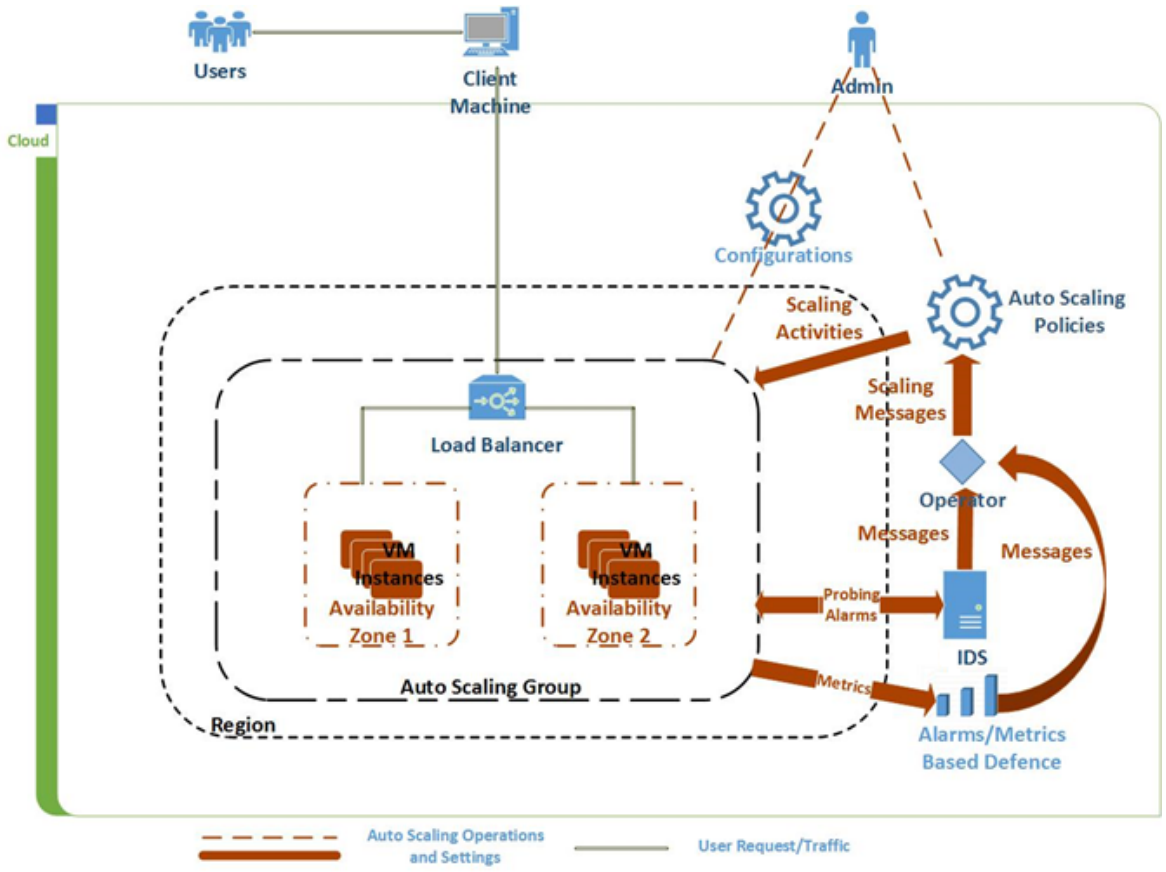


Figure 4.2 Proposed new architecture

average CPU utilization and HTTP load balancing serving capacity, which can be based on either utilization or requests per second.

Let n be the number of HTTP request in a VM time unit,
let u average CPU utilization during that time,
let $UF = n/u$ (Utilization Factor)

The Utilization Factor will be used to make a decision on whether there is legitimate increase in load due to more HTTP requests or the increase in load is due to other reasons. Given equal u, higher UF means several request are being serviced by VM which is potentially a good thing but low UF means few HTTP requests are occupying too many resources which signifies potential Application layer attack. This is assuming the volumetric based defenses have done their work, to avoid false negative in case of DDoS. Different applications will have different UF based on the requirements of the applications. Administrator have the role of assigning the UF based on past history and application needs. Other metrics can be used to achieve similar results.

4.5.2 Intrusion Detection System

Alternative way, albeit an expensive one, is the use a IDS that will analyze every packet coming into the VM to check the possibility of application layer attacks. Features to look at to detect include number of requests, HTTP header inspection, content-length, number of elements, nesting depth, longest element and name space in a SOAP Message. The possible attacks that will be looked at include flooding, Header outlier, size outlier, Feature outline and Coercive parsing. Tradition IDS such as Snort [74] have established signatures and way to generate alarms when intrusion is detected. Figure 4.2 shows the location of both metrics and the IDS based approaches in the cloud architecture.

4.6 Assessment

This section provides the comparison between the current and proposed architectures based on the type of attack that occurs and the speed of auto-scaling decision. To have a better comparison there is a need to define a threshold for normal behavior depending on application needs, to do that some sorts of datasets are needed. Unfortunately the dataset is not available, therefore, comparison of proposed and existing approaches was done based on some hypothetical scenarios. The summary of the comparison is that the proposed approaches are better alternatives than the current model in all the facets analyzed.

4.6.1 Application Layer DoS

Application Layer DoS occurs when a single packet causes exhaustion of server resources. This will normally affect a single VMs in the cloud environment.

a. Current Architecture

System will scale based on set parameters. In this case if the increase in load is as a result of application layer DoS attack not genuine customer requests, cloud customer will pay for resources that they do not need. This is because the auto-scaling policies module of Figure 4.1 will be used.

b. Proposed Architecture

System scale based on set parameters from the auto-scaling policies section of Figure 4.2. The customer will be aware of the reason why scaling occurred. Provider will also be aware of the reason for scaling, will be able to stop future attacks, and will scale the system down

to the appropriate resources. These will be reported by Alarms/Metrics based defense and detected by IDS modules of Figure 4.2.

4.6.2 Volumetric DoS (Flooding)

Volumetric DDoS occurs when the resources of servers are increased due to increase in malicious network traffic. It is sometimes distributed because the request are sent by multiple compromised nodes across the Internet. This attack has the potential to affect multiple VMs clusters or even cloud gateways depending on their severity.

a. Current Architecture

Scaling will occur as defined by the rules. Here also, if the increase in load is as a result of Application Layer DoS attack not genuine customer requests, cloud customer will pay for resources that they do not need. This is because the auto-scaling policies module of Figure 4.1 will be used.

b. Proposed Architecture

Scaling will occur as defined by the rules but customer will be aware of the reason why scaling occurred. Provider will also be aware of the reason for scaling, attack can be stopped, therefore scaling system down to the appropriate load and compensate customer. reported by Alarms/Metrics based defense and detected by IDS modules of Figure 4.2.

4.6.3 Speed of Auto-Scaling

This is a performance based case where how quickly auto-scaling decision can be made when load increase reaches a predefined threshold is checked.

a. Current Architecture

Fast based on pre-defined rules because this can be checked in constant time relative to the length of the packet. This is because the auto-scaling policies module of Figure 4.1 will be used and rules can be checked in constant time.

b. Proposed Architecture

The metrics based system is equally fast based on simple arithmetic of predefined rules and decision can be made in constant time. However, the IDS based system is slower because

entire packets have to be check. This is generally linear to the size of the message. The detection is usually more expensive than applying simple rules, this is done by IDS module of Figure 4.2.

4.6.4 Resource Over-utilization

Increase in load (resource over-utilization) can be genuine increase in business activity, application Layer DoS or Volumetric DDoS.

a. Current Architecture

Resources are simply increased without taking into account why there is an increase in the load. Again auto-scaling policies module of Figure 4.1 will be used.

b. Proposed Architecture

Resources are added and system investigates whether the increase is due to a legitimate increase in load or as a result of application layer attacks. Reported by Alarms/Metrics based defense module Figure 4.2.

4.7 Summary

Cloud computing currently supports many information systems and it will continue to be used. However, it is very important to ensure that both the customers and the vendors get their fair share of compensation in the presence of increasing hostility from attackers. This chapter introduced scaling techniques and the limitations of scaling techniques for current applications with ever changing workloads. Most current cloud service providers offer *auto-scaling* mechanisms that are better suited for today's dynamic environment, but most cloud providers support auto-scaling techniques that are reactive and they do not investigate the reasons where applications fail or why there is increased load in their VMs. A mechanism for checking the cause of failure was then proposed. This mechanism is based on the currently available metrics that are already provided to customers, while another approach is to use IDS to detect DoS at the nodes (VMs) in the cloud environment. The future work of this research to test the proposed model and have empirical results to reiterate the advantages as discussed in the initial assessment. Another possible direction is to look other combination of metrics because only one was discussed in this chapter.

Chapter 5

Reliable Job Scheduling in Cloud Computing - Centralized Approaches

5.1 Overview

While there is a growth in the use of cloud services, many potential users are still reluctant to deploy their business in the cloud. Major concerns are its reliability, security and stability [22]. There are different reliability and security issues depending on different delivery models of cloud services, including *SaaS*, *PaaS* and *IaaS*. This work focuses on the IaaS model. In particular, the scenarios when part of the workload is unreliable, e.g., fault-prone or generated by malicious sources, and a lightweight framework that combines load management and detection of unreliable traffic is proposed. This work investigates how to strike a balance between efficient workload scheduling and packet/job scanning so that stability can be maintained (as a guarantee of bounded buffers at machines) without sacrificing too much resources to filter out the unreliable part of the workload.

IaaS provides users with computing infrastructure in the form of VM. Following [15], assume that the users request resources such as memory, CPU and storage, for a certain amount of time in the form of VMs; this corresponds to a job to be done. Upon receiving the requests (typically in a form of packets), the system has to allocate the required resources by scheduling the VMs on the server. The model in [15] is extended by considering scenarios where part of the workload is genuine and the other unreliable. Genuine traffic comes from real users; completing these requests counts towards system's work done. Unreliable traffic is subject to failures or comes from attackers, who aim to disrupt the system by issuing requests that occupy resources; completing these does not count as proper work done. Classic reliability and security tool of packet scanning to detect these malicious packets [14, 74] was

adopted. While scanning is able to distinguish genuine from unreliable requests, it consumes resources that would normally be used for serving genuine workload. On the other hand, it is not known whether the packets are faulty/malicious until they are scanned, time and resources may be wasted in scanning genuine packets. Therefore, the scheduling algorithm needs to strike a balance between the resources wasted by scanning and by performing unreliable requests without scanning them.

Centralized scheduling algorithms are considered in this chapter, decentralized algorithms will be discussed in the next chapter. In the *centralized setting*, there are central queues, and upon arrival jobs are added to the central queues corresponding to the requested type of VMs; recall that there is a limited number of types of VMs (as each of VMs is in fact a small operating system [15]) and each job is allocated to a VM of the requested type. When the resources become available, the centralized scheduling algorithm determines which set of jobs is to be served and to which servers the VMs are mapped to.

The system is *stable* if the queues do not tend to increase without bound. The aim is to characterize the maximum arrival rates of genuine and unreliable requests under which there exists an algorithm to maintain the stability of the system and to develop such algorithm if it exists. In addition, to guarantee quality of service, job latency is measured, which is defined as the amount of time a job resides in the system since its arrival.

The precise model is presented in Section 5.2, the proposed algorithm is described in Section 5.3 and theoretical analysis in 5.4 and 5.5. Simulations and conclusions are presented in Sections 5.6 and 5.7 respectively.

5.2 Cloud Computing Model

Consider a cloud system modeled by a network of physical machines that have limited available resources (for instance, CPU, memory, storage, ...) and is supposed to be able to process an ongoing stream of jobs.

Servers. Consider a set of n networked servers (physical machines). Each server has its own resources that it can distribute among jobs, that is, for each resource it has a fixed capacity.

Jobs. A job is specified by its type and length. Since there are limited number of virtual machine types, only limited number of job types are considered– there are J types of jobs.

Each type is a set of demands for resources; more specifically, for each available server resource a job type has a number specifying how much of this resource is required in order to process any job of this type.

There are I different lengths of jobs possible: $L_1, \ldots, L_I$ Online random arrival model is considered, where new jobs arrive independently of each other and are identically distributed across all time slots, and the variance of arrival length is finite. Let $\lambda_{i,j}$ denote expected sum of lengths of genuine (i.e., user-generated) type-j jobs of length L_i that arrive per time slot, for any positive integers $j \leq J$ and $i \leq I$.

Processing jobs and feasible configurations. Each server can process a set of jobs simultaneously, as long as the cumulative amount of each resource used by these jobs

does not exceed the server capacity for this resource. Processing jobs is done in synchronous time steps, also called rounds. The whole system capacity is a linear sum of capacities of all the servers. Given job types and server capacities, one can compute the set $\mathcal{S}$ of all feasible configurations, where feasible configuration denotes a vector $N = (N_1, \ldots, N_J)$ such that the system can process simultaneously N_j type-j jobs, for every j. For example, consider a server with 30 GB memory, 30 EC2 computing units and 4000 GB storage space. If arriving jobs are served in the cloud based on three types of virtual machines described in Table 5.1 then, this gives three feasible (maximal) configurations available at each server: $(2,0,0)$, $(1,0,1)$ and $(0,1,1)$.

Table 5.1 Representation of Instances in Amazon EC2

Instance type	Memory (GB)	vCPU	Storage (GB)
Standard	15	8	1,690
High-Memory	17.1	6.5	420
High-CPU	7	20	1,690

Malicious jobs and security tools. Let $\kappa_{i,j}$ denote the expected sum of lengths of malicious jobs of type-j of length L_i that arrive per time slot. Similarly as genuine jobs, malicious jobs arrive independently of each other and are identically distributed across all time slots, and the variance is finite. Assume that there is a scanning tool that, given a job, can detect whether it is a genuine user request (call it a *good job*) or a malicious request (a *malicious job*). Each scanning takes 1 time slot per job and requires same resources as the original job (scanning is done on the same virtual machine).

Central scheduler. Consider a central scheduler with a queue of all injected, but not yet finished jobs. The scheduler decides which servers process which jobs for the next time slot. After this time slot, all unfinished jobs return to the scheduler with saved progress and can be processed further at a later time and by a different server. This property of a system is called preventiveness. The centralized algorithm, called SecureMaxWork, will be introduced in Section 5.3.

In the decentralized approach each server runs locally a protocol SecureMaxWork with respect to its local queues. The decentralized implementations of algorithm SecureMaxWork will be presented in the next chapter.

Notation.

- n denotes the number of servers in the cloud;
- I denotes the number of different job lengths;
- J denotes the number of different job types;
- $A(t) = (A_1(t), \ldots, A_J(t))$ denotes the vector of sets of type-j jobs, for $j \leq J$, which arrive to the system in the beginning of time slot t;
- $Q(t)$ denotes the vector of queue lengths (i.e., sum of lengths of jobs in the queue) for each type of jobs in the beginning of time slot t;
- $Q_j(t)$ denotes the total length of users' and malicious type-j jobs, for $j \leq J$, in the beginning of time slot t;

In addition there are the following notations regarding SecureMaxWork algorithm:

- $\alpha_{i,j}$ is a probability of scanning type-j job of length L_i; the algorithm may implement a specific scanning strategy, i.e., use a specific vector α.
- $X_j(t)$ is the total length of queued type-j jobs that will not be scanned, taken in the beginning of time slot t (i.e., the algorithm scanned them already or decided not to scan them at all);
- $Y_j(t)$ is the total length of queued type-j jobs that will be scanned, taken in the beginning of time slot t (i.e., the algorithm has already decided to scan them, but has not scanned them yet);
- $Z_j(t) = Z_j(Q(t))$ is the expected time required to process type-j jobs stored in queue in the beginning of time slot t; the formula with an explanation for it will be given in section 5.3;
- a_j is the expected time required to process type-j jobs that arrive in one time slot; the formula with an explanation for it will be given in section 5.5;
- a denotes the vector of expected time required to process all jobs that arrive in one time slot;

- ℓ_j is a (random) length of arriving type-j jobs.

Whenever time slot t is clearly fixed or understood from the context, argument t may be omitted from the formulas.

Scanning strategies. The following scanning strategies will be compared:

- ScanNONE — always executes a job without scanning, i.e., $\alpha_{i,j} = 0$ for all i, j;
- ScanALL — scans all jobs except those with processing time shorter or equal to the scanning time (recall that scanning takes 1 time slot), i.e., $\alpha_{i,j} = 0$ for $L_i \leq 1$ and $\alpha_{i,j} = 1$ otherwise;
- ScanOPT — will be defined in section 5.5.2.

Stability. Given arrival rates λ and κ, the algorithm is stable if the expected queue size at any fixed time is bounded, i.e. $\limsup_{t\to\infty} E[\sum_j Q_j(t)] < \infty$.

5.3 Main Algorithm

Algorithm SecureMaxWork (Algorithm 1, with Algorithms 2, 4 and 3 as sub-procedures) is parametrized by: scanning vector $\alpha = (\alpha_{i,j})_{i\leq I, j\leq J} \in [0,1]^{I\times J}$, vector of rates of genuine user's requests $\lambda = (\lambda_{i,j})_{i\leq I, j\leq J}$, and vector of rates of unreliable requests $\kappa = (\kappa_{i,j})_{i\leq I, j\leq J}$. Upon arrival of type-j job of length L_i, SecureMaxWork decides to scan it with probability $\alpha_{i,j}$ (c.f., the first **for all** loop in Algorithm 1).

The key idea of SecureMaxWork is to measure the expected time required to process all jobs of each type and prioritize the type which accumulates the most. The expected time (also called expected work) required to process all jobs of type j accumulated in queue at time t is denoted by $Z_j(t)$.

It takes X_j time to process jobs that will not be scanned. Jobs contributing to Y_j will need to be scanned, requiring $Y_j \cdot E(1/\ell_j)$ expected time. In expectation $\lambda_j/(\lambda_j+\kappa_j)$ fraction of scanned jobs are genuine, so after scanning, they still must be processed, taking in total $Y_j \cdot \lambda_j/(\lambda_j+\kappa_j)$ time. $\kappa_j/(\lambda_j+\kappa_j)$ fraction of scanned jobs are malicious and after scanning they take no more processing time. Therefore: $Z_j(t) = X_j(t) + Y_j(t) \cdot (\lambda_j/(\lambda_j + \kappa_j) + E(1/\ell_j))$.

The algorithm then computes Z_j (c.f., the second **for all** loop in Algorithm 1) and finds configuration N from the set of feasible server configurations $\mathcal{S}$ that maximizes the sum $\sum_{j=0}^{J} Z_j(t)N_j$, i.e., the objective in each time slot t is:

$$\max_{N\in S} \sum_{j=0}^{J} Z_j(t)N_j \ . \tag{5.1}$$

Algorithm 1 SecureMaxWork(λ, κ, α)

```
X ← 0⃗   // jobs that will not be scanned
Y ← 0⃗   // jobs that will be scanned
Q ← 0⃗   // all jobs
loop
    new time slot begins
    new jobs arrive
    for all new type-j job τ_{i,j} of length L_i do
        r ← random value from [0;1]
        if r < α_{i,j} then                          // τ_{i,j} to be scanned
            Y_j ← Y_j + L_i
            Q_j ← Q_j + L_i
        else                                         // τ_{i,j} not to be scanned
            X_j ← X_j + L_i
            Q_j ← Q_j + L_i
        end if
    end for
    for all j do
        Z_j ← X_j + Y_j(λ_j/(λ_j + κ_j) + E(1/ℓ_j))
    end for
    N' ← arg max_{N∈S} Σ_j N_j · Z_j
    for all j do
        for k ≤ N'_j do
            Process_job(j)
        end for
    end for
end loop
```

This configuration is denoted by N'. The intuition is that the more jobs of a given type accumulate, the more weight should be put to scheduling this job type to prevent further accumulation. Z_j here is the weight given to jobs of type j.

Finally, in the last **for all** loop, the algorithm processes N'_j jobs of type j, for each $1 \leq j \leq J$; i.e., from each job processed it executes a unit of it and the total size of Q_j decreases by N'_j at the end of time slot t. It is done by calling procedure Process_job(j), c.f., Algorithm 2. If N'_j is larger than the number of different type-j jobs in the queues, SecureMaxWork processes as many type-j jobs as possible instead, each time processing a unit of each such job. If N'_j is smaller than the number of different type-j jobs in the queues, SecureMaxWork has to decide which type-j jobs to process. It repeats N'_j times:

- with probability $X_j/(X_j+Y_j)$ it processes a job that will not be scanned (i.e., a job that contributes to X_j),

Algorithm 2 Process_job(j)

if there are still jobs in X_j and Y_j that are not yet scheduled to be processed in this time slot **then**
  $r \leftarrow$ random value from $[0;1]$
  if $r < X_j/(X_j+Y_j)$ **then**
    Process_job_X(j)
  else
    Process_job_Y(j)
  end if
else if there are no jobs in X_j that are not yet processed in this time slot, but there are still such jobs in Y_j **then**
  Process_job_Y(j)
else if there are no jobs in Y_j that are not yet processed in this time slot, but there are still such jobs in X_j **then**
  Process_job_X(j)
end if

Algorithm 3 Process_job_Y(j) // to be scanned

Scan any unscheduled job contributing to Y_j
$L_i \leftarrow$ length of the scheduled job
$Y_j \leftarrow Y_j - L_i$
if detected as unreliable **then**
  $Q_j \leftarrow Q_j - L_i$
else
  $X_j \leftarrow X_j + L_i$
end if

Algorithm 4 Process_job_X(j) // not to be scanned

Process a unit of any unscheduled job contributing to X_j
$X_j \leftarrow X_j - 1$
$Q_j \leftarrow Q_j - 1$

- with probability $Y_j/(X_j+Y_j)$ it scans a job pending for scanning (i.e., a job that contributes to Y_j).

If there are not enough jobs contributing to X_j, it processes all jobs contributing to X_j and as many jobs contributing to Y_j as possible, so that altogether it processes N_j type-j jobs (vice versa for Y_j). Processing and/or scanning a specific type-j job is done by calling sub-procedures Process_job_X(j) and/or Process_job_Y(j), resp. (c.f., Algorithms 4 and 3, resp.).

5.4 Analysis

This section proves that algorithm SecureMaxWork is stable if there is $\varepsilon > 0$ and vector a such that $(1-\varepsilon)\cdot a \in co(\mathcal{S})$, where $co(\mathcal{S})$ denotes convex hull of set $\mathcal{S}$.

Theorem 1 *The SecureMaxWork algorithm is stable for all arrival patterns λ, κ, for which there exists a stable algorithm.*

In the remainder of the section proof of Theorem 1 will be provided. The following results are needed (extension of Foster's criteria for irreducible Markov chains).

Theorem 2 ([90]) *Consider a Markov chain $Q(t)$ with state space Ω. Consider a random walk on it, starting from a state x. Let τ_x denote the time when the random walk first reaches some recurrent state (or infinity if it never reaches any).Let R_j be a closed set of communicating states and T be set that contains all the states no in R_j. If there exists a lower bounded real function $V : \Omega \to \mathbb{R}$, an $\varepsilon > 0$ and a finite subset Ω_0 of Ω such that*

$$E[V(Q(t+1)) - V(Q(t))|Q(t) = q] < -\varepsilon, \quad \text{if } q \notin \Omega_0, \tag{5.2}$$

$$E[V(Q(t+1))|Q(t) = q] < \infty, \quad \text{if } q \in \Omega_0, \tag{5.3}$$

then

$$P(\tau_q < \infty) = 1, \quad \forall q \in T \tag{5.4}$$

and all states $q \in \cup_{j=1}^{\infty} R_j$ are positive recurrent.

Let $V(Q(t)) = \sum_j (Z_j(Q(t)))^2$. Note that $V(Q(t)) \geq 0$ for all possible queue states $Q(t) \in \Omega$. It will be shown that there exist two positive numbers b, ε such that the inequality

$$E[V(Q(t+1)) - V(Q(t))|Q(t) = q] < -\varepsilon \tag{5.5}$$

holds for all $q \in \Omega$ for which there exists $q_j \geq b$.

Let $A(t)$ and $Alg(t)$ denote the vector of arrival lengths and (respectively) the vector of queue changes due to algorithm decisions – for each type of job, with distinction between jobs that will be scanned and jobs that will not be scanned, in the beginning of time slot t.

Z_j will be used as a shorthand of $Z_j(Q(t))$, A as a shorthand of $A(t+1)$, and Alg as a shorthand for $Alg(t+1)$.

$$\begin{aligned} & E[V(Q(t+1)) - V(Q(t))|Q(t) = q] \\ &= E[\sum_j [Z_j(Q(t+1))^2 - Z_j^2]|Q(t) = q] \\ &= E[\sum_j [(Z_j + Z_j(A) - Z_j(Alg))^2 - Z_j^2]|Q(t) = q] \\ &= E[\sum_j [(Z_j(A) - Z_j(Alg))^2 + \\ &\quad +2Z_j(A - Z_j(Alg))]|Q(t) = q] \\ &\leq K + 2E[\sum_j [Z_j \cdot Z_j(A)]|Q(t) = q] + \\ &\quad -2E[\sum_j [[Z_j \cdot Z_j(Alg)]|Q(t) = q] \ . \end{aligned} \tag{5.6}$$

The last inequality comes from $E[\sum_j [(Z_j(A) - Z_j(Alg))^2]|Q(t) = q]$ being upper bounded under assumption that the variances of arrival lengths are finite; this upper bound is denoted by K.

Lemma 1 *There exists a finite set $\mathcal{F} \subseteq \mathcal{Q}$ such that for all $q \in \mathcal{Q} - \mathcal{F}$:*

$$\begin{aligned} K + 2E[\sum_j [Z_j \cdot Z_j(A)]|Q(t) = q] + \\ -2E[\sum_j [[Z_j \cdot Z_j(Alg)]|Q(t) = q] < 0 \ . \end{aligned}$$

To prove Lemma 1, the following result is needed:

Lemma 2 *For almost all queue states q there exists a feasible configuration $N \in S$ such that $N \cdot Z \geq K + a \cdot Z$ (where $Z = Z(q)$ and $\cdot$ is scalar product).*

Proof sketch of Lemma 2: Vector $(1+\varepsilon)a$ lies inside convex hull of set $\mathcal{S}$ (set $\mathcal{S}$ is the set of feasible server configurations N). Therefore vector a is smaller (and not equal) than some linear combination of vectors from $\mathcal{S}$. So for any non-negative vector Z the following inequality holds: $N \cdot Z > a \cdot Z$. So if any large enough vector Z is taken, then $N \cdot Z > a \cdot Z + K$. There is only a finite number of queue states q that generate not large enough vectors Z. ■

Proof of Lemma 1: Note that in Lemma 1, $E[\sum_j [Z_j \cdot Z_j(A)]]$ is the expected value of the scalar product $Z \cdot Z(A)$ and $E[\sum_j [[Z_j \cdot Z_j(Alg)]$ is the expected value of the scalar product $Z \cdot Z(Alg)$. According to Lemma 2, inequality from lemma 1 is true for almost all queue states, i.e., there exists a finite set of states $\mathcal{F}$ such that for all $q \in \mathcal{Q} - \mathcal{F}$ the desired inequality holds. ■

Proof of Theorem 1: According to Lemma 1 and Theorem 2, given arrival rates for which there exists a stable algorithm, SecureMaxWork algorithm reaches positive recurrent state in a finite time, therefore it is stable. ■

5.5 Determining Feasible Capacity Region and Scanning Frequency

5.5.1 Feasible Capacity Region

Lemma 3 *Processing a type-j job for 1 time slot decreases $Z_j(Q(t))$ by 1 on average.*

Proof: If the processed step was not scanning (i.e., processing a job from X_j) then trivially Z_j decreased by 1.

If the processed step was scanning a job of length L_i (i.e., processing a job from Y_j), then:

- before scanning that job contributed $\frac{\lambda_j}{\lambda_j+\kappa_j} \cdot L_i + 1$ weight towards Z_j;
- with probability $\frac{\lambda_j}{\lambda_j+\kappa_j}$ it was a genuine job, so after scanning it contributes L_i towards Z_j (increase in weight);
- with probability $\frac{\kappa_j}{\lambda_j+\kappa_j}$ it was a malicious job, so after scanning it contributes 0 towards Z_j (decrease in weight).

Therefore, on average Z_j decreases by

$$\begin{aligned} &\tfrac{\lambda_j}{\lambda_j+\kappa_j}(1-\tfrac{\kappa_j}{\lambda_j+\kappa_j}L_i)+\tfrac{\kappa_j}{\lambda_j+\kappa_j}(1+\tfrac{\lambda_j}{\lambda_j+\kappa_j}L_i) = \\ &\quad = \; 1+\tfrac{\lambda_j\kappa_j}{(\lambda_j+\kappa_j)^2}(-L_i+L_i) = 1\ . \end{aligned} \tag{5.7}$$

■

Recall that $\alpha_{i,j}$ is the probability of scanning type-j job of length L_i, and $A(t)$ denotes the vector of arrival lengths for each type of job, with distinction between jobs that will be scanned and jobs that will not be scanned in the beginning of time slot t.

Let $a_j = a_j(\alpha,\lambda,\kappa) = E[Z_j(A(t))]$ be the expected weight of type-j jobs that arrive per time slot (arrivals are i.i.d. across time slots, so $E[A_j(t)] = E[A_j(t+1)]$ for all t). Then

$$a_j = \sum_i p_{i,j}[(\lambda_j+\kappa_j)(1-\alpha_{i,j}) + \alpha_{j,i}(\lambda_j+\kappa_j)(\frac{\lambda_j}{\lambda_j+\kappa_j}+\frac{1}{L_i})], \tag{5.8}$$

where $p_{i,j}$ is the probability that type-j job has length L_i. Addend $(\lambda_j+\kappa_j)(1-\alpha_{i,j})$ corresponds to the weight of good and malicious jobs that will not be scanned (so it contributes to X_j). Addend $\alpha_j(\lambda_j+\kappa_j)\cdot\frac{\lambda_j}{\lambda_j+\kappa_j}$ corresponds to the weight of good jobs that will be scanned but without scanning taken into account yet (so it contributes to Y_j). Addend $\alpha_j(\lambda_j+\kappa_j)\cdot 1/L_i$ corresponds to the weight of scanning good and malicious jobs (so it also contributes to Y_j). Let $a = (a_1,\ldots,a_J)$.

Theorem 3 *If arrivals λ and κ are such that for all vectors α arrivals $a \notin co(\mathcal{S})$ then no algorithm is stable.*

Proof: Consider arrival rates λ, κ and scanning probabilities α such that $a \notin co(\mathcal{S})$. It will be shown that for every algorithm there exists j such that $E[Z_j(Q(t))]$ is unbounded.

In every time slot t the weight of queues $Z(Q(t))$ is changing on average by $E[Z(Q(t+1)) - Z(Q(t))] = E[Z(Q(t) + A(t+1) - Alg(t+1)) - Z(Q(t))] = E[Z(A(t+1)) - Z(Alg(t+1))] = a - E[Z(Alg(t+1))]$.

Note that $a \notin co(\mathcal{S})$, while $E[Z(Alg(t+1)] = N(t+1) \in co(\mathcal{S})$ (according to Lemma 3). If multiple time slots are considered and any combination of algorithm decisions $N(t)$ then the vector of weights of queues $Z(Q(t))$ is growing in the direction of vector a. Therefore there exists j such that $Z_j(Q(t))$ is unbounded with regard to t. Therefore $Q_j(t)$ is unbounded with regard to t, which is contradictory with the definition of stability. ■

5.5.2 Optimal Scanning Frequencies

Theorem 4 *If there exists a vector of scanning frequencies $\alpha^{(0)}$ such that given job arrival rates λ and κ are inside the capacity region (i.e., $(1+\varepsilon)a(\alpha^{(0)}, \lambda, \kappa) \in co(\mathcal{S})$ as defined in subsection 5.5.1), then there exists a vector of scanning frequencies $\alpha^{(1)} \in \{0,1\}^{I \times J}$ such that these job arrivals are inside the capacity region ($(1+\varepsilon)a(\alpha^{(1)}, \lambda, \kappa) \in co(\mathcal{S})$).*

Proof: Function $a_j(\alpha, \lambda, \kappa)$ (as defined in subsection 5.5.1) is independent of $\lambda_k, \kappa_k, \alpha_{i,k}$ for $k \neq j$ and for all i. Given fixed λ and κ, $a_j(\alpha, \lambda, \kappa)$ is a linear combination of scanning frequencies $\alpha_{i,j} \in [0,1]$, for all i.

Therefore the vector $\alpha_j = (\alpha_{1,j}, \alpha_{2,j}, \ldots, \alpha_{I,j})$ that minimizes a_j is one of the extreme points of region $[0,1]^I$. Furthermore, $\alpha_{i,j}^{(1)}$ for all i that minimize a_j can easily be computed, since each summand $p_{i,j}[(\lambda_j + \kappa_j)(1 - \alpha_{i,j}) + \alpha_{i,j}(\lambda_j + \kappa_j)(\frac{\lambda_j}{\lambda_j + \kappa_j} + 1/L_i))]$ is independent of all other summands. Value $\alpha_{i,j}^{(1)}$ that minimizes this summand is 0, if $1 \leq \frac{\lambda_j}{\lambda_j + \kappa_j} + 1/L_i$, and 1 if $1 > \frac{\lambda_j}{\lambda_j + \kappa_j} + 1/L_i$, for each i, j.

This gives $a(\alpha^{(0)}, \lambda, \kappa) \geq a(\alpha^{(1)}, \lambda, \kappa))$, which means that $(1+\varepsilon)a(\alpha^{(1)}, \lambda, \kappa)) \in co(S)$. ■

5.6 Simulations

5.6.1 Experiment Setting

The setup for simulations, described in this section, is based on the one in Maguluri et al. [15].

Servers and VMs Servers are based on the configuration described in Section 5.2 where processing jobs and feasible configurations were described. Consider 100 identical servers in the cloud. The VMs are the same as the ones in Table 5.1.

Job arrivals The generic arrival vector $\lambda^* = 0.99 \times (1, 1/3, 2/3)$ for the genuine users' workload was used, this is located at the border of the server capacity area (it is easy to observe that it is a normalized linear combination of the three maximal configurations, additionally re-scaled by factor 0.99). In each time step a job of type $j = 1, 2, 3$ is selected with probability $\frac{\lambda_j^*}{130.5}$, and its length is chosen according to the length distribution described below with the mean length 130.5.

Similarly as above, malicious workload defined by a generic arrival vector $\kappa^* = (0.7, 0.01, 0.01)$, and the procedure of generating a malicious traffic is analogous as above for generating the genuine users' one. Note that each of the arrival rates λ^* and κ^* is within the capacity range of a server, whereas the combined work-flow rate $\lambda^* + \kappa^*$ is not.

Job size distribution When a new job is generated, with the probability of 0.7 it is an integer that is uniformly distributed in the interval $[1, 50]$, with the probability of 0.15 it is an integer uniformly distributed in the interval $[251, 300]$, and with the probability of 0.15 it is an integer uniformly distributed in the interval $[451, 500]$. Note that there are 150 possible job lengths, and the mean length is 130.5, as assumed in the definition of arrival rates.

Setup of simulations. Since there are 100 homogeneous servers, the overall arrival rates are: $\lambda = 100 \times \lambda^* = (99, 33, 66)$ for genuine workload, and $\kappa = 100 \times \kappa^* = (70, 1, 1)$ for malicious workload. The job size distribution is as specified above, same for each job type. The following optimal scanning vector α^* is computed for this setting, more precisely, the vector minimizing expected arriving weight:

- $\alpha_{i,1}^* = 0$ for $L_i \leq 2$,
- $\alpha_{i,2}^* = 0$ for $L_i \leq 34$,

- $\alpha^*_{i,3} = 0$ for $L_i \leq 50$,
- $\alpha^*_{i,j} = 1$ otherwise.

Each execution includes $4,000,000$ time steps. The *average latency*, *maximum latency*, *average queue size* and *maximum queue size* were computed at at every time step and the results were recorded at every $200,000$ steps. The experiments were ran 10 times and the averages of the results for each recorded time step was recorded.

LambdaFlow: SecureMaxWork applied for genuine flow only, (i.e., only with genuine arrival rate λ), and no scanning is applied (i.e., $\vec{\alpha} = \mathbf{0}$);

ScanOPT: SecureMaxWork applied for simultaneous genuine and malicious flows, with scanning defined by vector α^*;

ScanALL: SecureMaxWork applied for simultaneous genuine and malicious flows, with scanning all jobs of size bigger than 1 (i.e., for every $j = 1,2,3$, $\vec{\alpha_{1,j}} = 0$ and $\vec{\alpha_{i,j}} = 1$ for every $i > 1$);

ScanNONE: SecureMaxWork applied for simultaneous genuine and malicious flows, with no scanning (i.e., $\vec{\alpha} = \mathbf{0}$).

As by theoretical part, it is expected that the first two executions should be stable while the last one is not. It is expected that the third execution is also not stable, which would justify the research quest for searching of suitable scanning vector. In order to visualize it, differences and ratios between the second and the third executions are also displayed — the stable and the potentially unstable one.

In the second part of the simulations, the arrival vector was varied by defining a parameter $\rho \in (0,1]$ called *traffic intensity*. Recall that the generic arrival vector $\lambda^* = 0.99 \cdot (1, 1/3, 2/3)$ for genuine traffic and malicious workload using a generic arrival vector $\kappa = (0.7, 0.01, 0.01)$ Here, the arrival vector λ^* and κ were varied and set $\rho \in \{0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 0.96, 0.97, 0.98, 0.99\}$. This means that the probabilities of the arrival of any of the three types of jobs reflect the different job intensities. In the first sets of experiment $\kappa = 1$ is fixed and λ^* is varied, this to observe how stability is affected by change in genuine workload. In the second part fixed λ was used and κ was varied to observe how change in unreliable traffic affects stability. Each execution includes $4,000,000$ time steps. *Average latency*, *maximum latency*, *average queue size* and *maximum queue size* were computed at every time step and values are recorded at the end of $4,000,000$ time steps. The experiments were ran 10 times and the averages of the results for each traffic intensity was recorded.

5.6.2 Performance Measure

The analysis for the queue stability of the algorithm was made. Stability relates to the average queue size. Therefore the average queue sizes of different algorithms were compared. Furthermore, measurement of maximal queue size, average latency and maximal latency was taken, these are also important in estimating efficiency of the algorithms. Large maximal queue size means that the system has to be able to find storage for these pending jobs when they happen to accumulate. Large average latency means that there is a large delay before jobs are finished – which is important in some applications. Large maximal latency means that the algorithm is inappropriate for real-time systems.

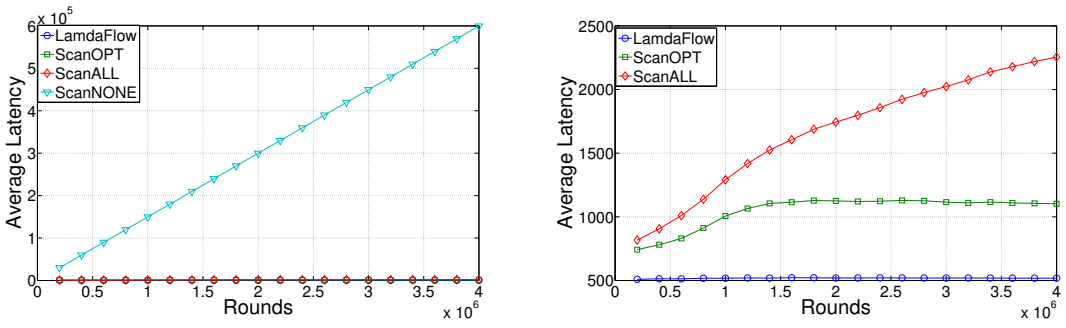
Note that maximal latency and maximal queue size grow to infinity as time increases for all algorithms. This is because the arrival rates considered are random and i.i.d. across time slots – so an event that for any time interval arrivals in each step of this interval exceed the arrival vector is non-zero. However the rate at which these measures grow to infinity depends on the load balancer used – some load balancers (such as JSW) are allowing the system to better utilize its resources and therefore remove jobs from the system faster.

5.6.3 Results of Simulations

Hypothesis 1 *Based on the theory, the ScanOPT strategy should be stabilizing.*

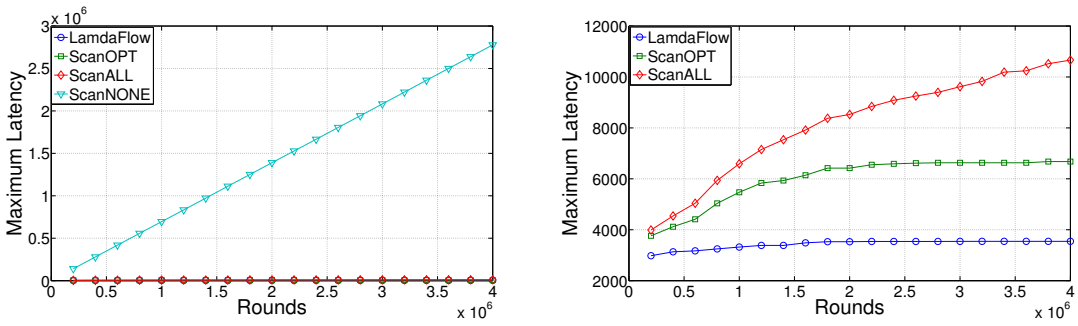
Hypothesis 2 *Based on the theory, the naive ScanALL will not be stabilizing.*

Hypothesis 3 *Based on the theory, the ScanNONE strategy should be increasing at an exponential rate as t increases due to increased backlog of jobs.*



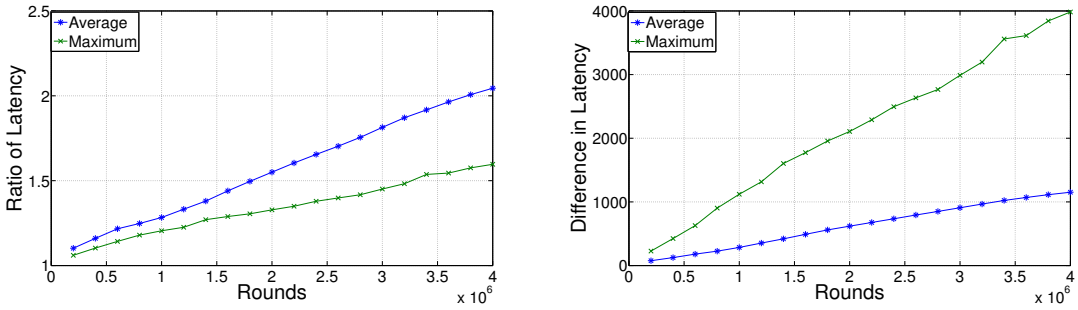
(a) (b) Zoomed version of (a)

Figure 5.1 Comparison of average latency of LambdaFlow, ScanOPT, ScanALL and ScanNONE strategies.



(a) (b) Zoomed version of (a)

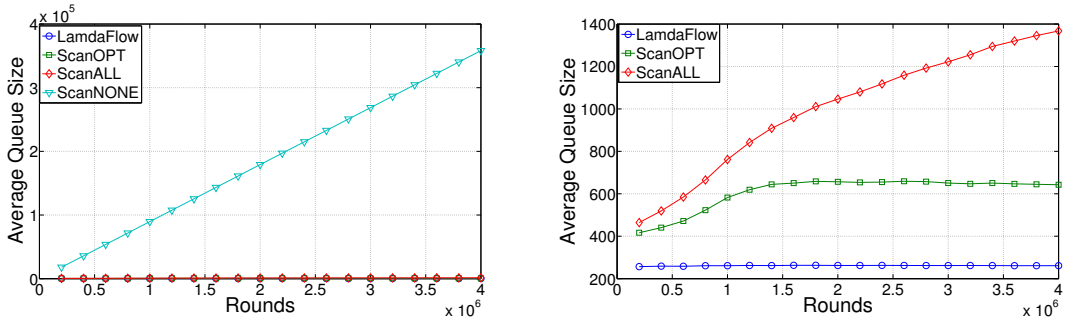
Figure 5.2 Comparison of maximum latency of LambdaFlow, ScanOPT, ScanALL and ScanNONE strategies.



(a) (b)

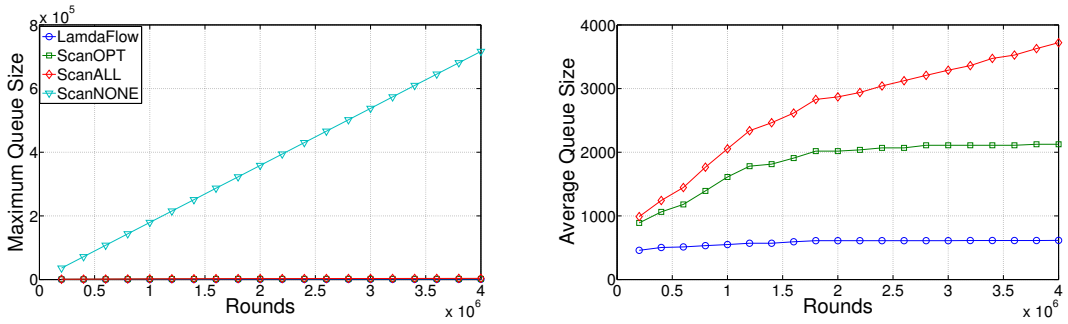
Figure 5.3 Ratio of ScanALL to ScanOPT latency and difference between ScanALL and ScanOPT latency (indicating ScanALL becomes worse over time).

In order to study throughput-optimality of the scanning strategies, the latency over time for the different scanning strategies used was recorded, i.e., ScanALL, ScanOPT and ScanNONE, comparing them with the execution LambdaFlow of the genuine workload only. In Figure 5.1, the average latency of the ScanNONE and ScanALL strategies grow rapidly, while it stabilizes for the ScanOPT strategy. The right part of the Figure is the zoomed left part, in order to see clearly the performance of ScanOPT versus ScanALL. The performance of ScanALL strategy is even worse for maximum latency, where it was observed that it increases rapidly; this is shown in Figure 5.2. This indicates that some jobs will eventually get stuck. In Figure 5.3 the ratio and the difference between ScanALL to ScanOPT latencies was analyzed, and both are increasing. This indicates that ScanALL is becoming worse over time, confirming the theory that ScanOPT stabilizes while ScanALL does not (c.f., Theorem 1 and Theorem 3, respectively, applied to the experiment setting of arrival rates λ, κ and scanning vectors α^* and ScanALL, respectively). This also affirms Hypotheses 1 , 2 and 3.



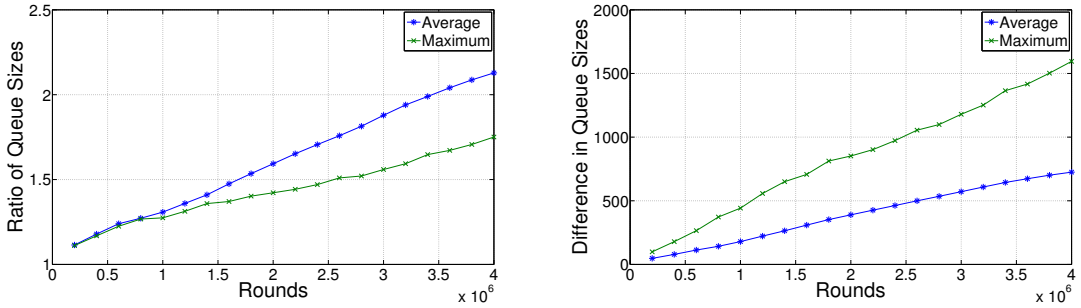
(a) (b) Zoomed version of (a)

Figure 5.4 Comparison of average queue sizes of LambdaFlow, ScanOPT, ScanALL and ScanNONE strategies.



(a) (b) Zoomed version of (a)

Figure 5.5 Comparison of maximum queue sizes of LambdaFlow, ScanOPT, ScanALL and ScanNONE strategies.

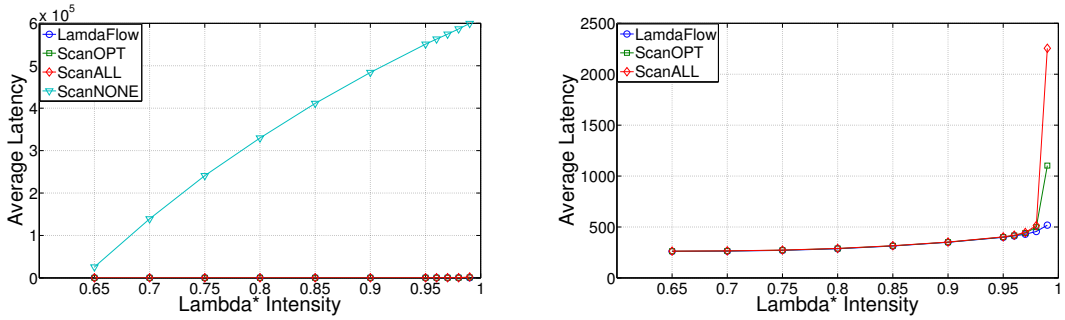


(a) (b)

Figure 5.6 Ratio of ScanALL to ScanOPT Queue sizes and difference between ScanALL and ScanOPT Queue Sizes.

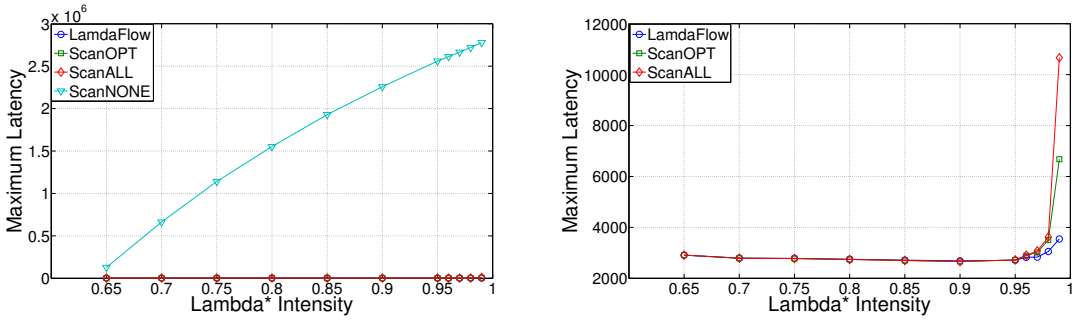
The figures measuring queue sizes over time show that the trend is in fact similar to the trend in latency with ScanALL performing considerably worse than ScanOPT while ScanNONE grow even more rapidly over time. Figure 5.4 shows the average while queue

sizes while Figure 5.5 shows the maximum queue sizes and Figure 5.6 shows the ratio and differences of ScanALL and ScanNONE increasing overtime.



(a) (b) Zoomed version of (a)

Figure 5.7 Comparison of average latency of LambdaFlow, ScanOPT, ScanALL and ScanNONE strategies with varying Intensity of λ



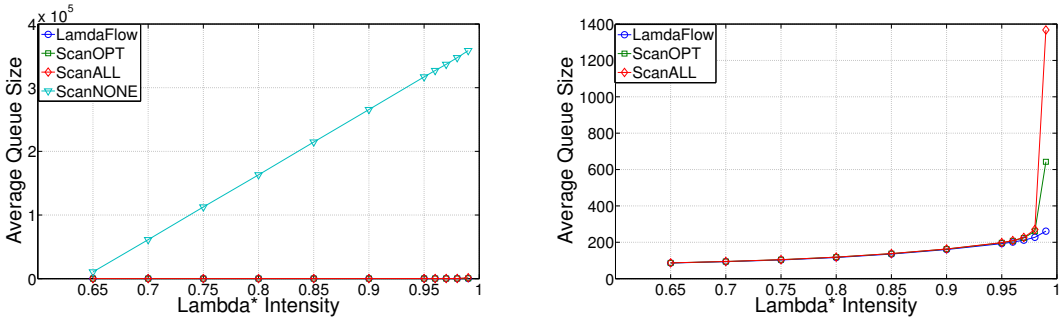
(a) (b) Zoomed version of (a)

Figure 5.8 Comparison of maximum latency of LambdaFlow, ScanOPT, ScanALL and ScanNONE strategies with varying Intensity of λ

Figures 5.7 and 5.8 compare the average and maximum latency of the three centralized approaches (ScanALL, ScanOPT and ScanNONE) and LambdaFlow. This is done while varying the intensity of genuine traffic. The trend is as expected where ScanNONE performs worst followed by ScanALL then the optimal ScanOPT. The queues were compared in Figures 5.9 and 5.10 where they show similar trend.

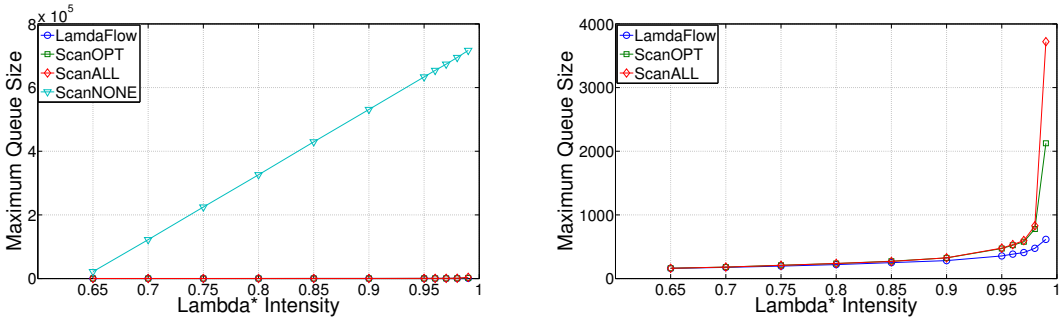
Hypothesis 4 *When κ is varied the ScanNONE should increase exponentially as κ intensity increases.*

Hypothesis 5 *The performance of ScanOPT is expected to be better than ScanALL for varying κ intensities as with the previous Hypothesis*



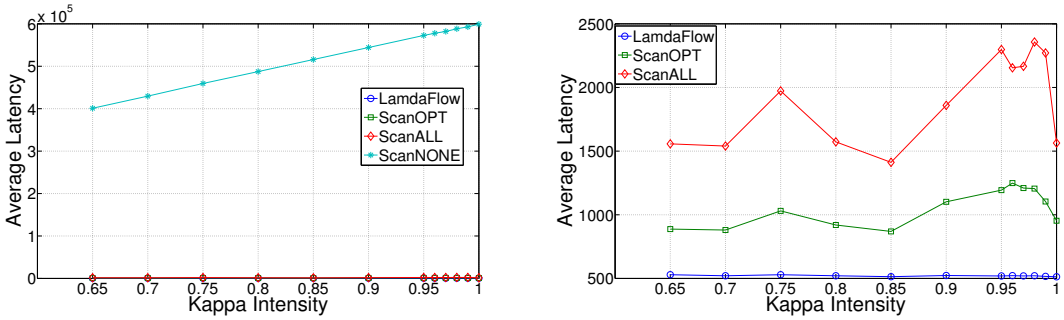
(a) (b) Zoomed version of (a)

Figure 5.9 Comparison of average queue sizes of LambdaFlow, ScanOPT, ScanALL and ScanNONE strategies with varying Intensity of λ



(a) (b) Zoomed version of (a)

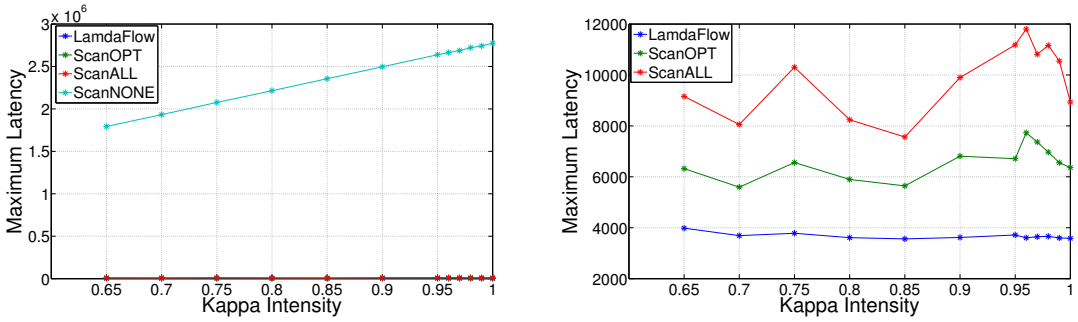
Figure 5.10 Comparison of maximum queue sizes of LambdaFlow, ScanOPT, ScanALL and ScanNONE strategies with varying Intensity of λ



(a) (b) Zoomed version of (a)

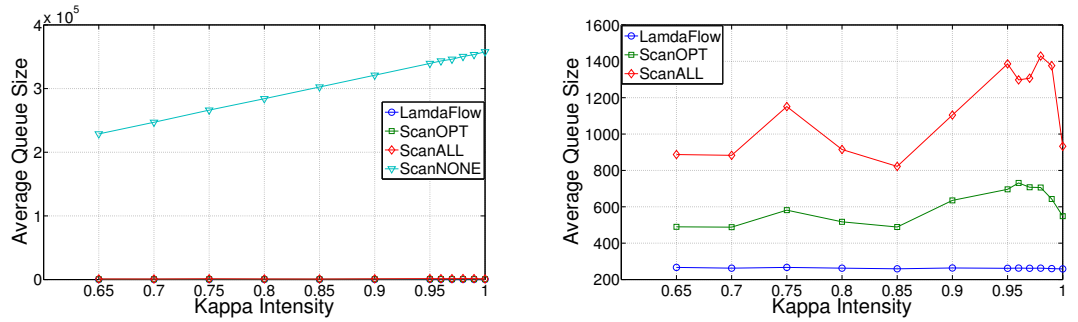
Figure 5.11 Comparison of average latency of LambdaFlow, ScanOPT, ScanALL and ScanNONE strategies with varying Intensity of κ

Figures 5.11 and 5.12 compare the average and maximum latency of the three centralized approaches (ScanALL, ScanOPT and ScanNONE) and LambdaFlow.The experiment is conducted while varying the intensity of unreliable traffic κ. The trend is as expected



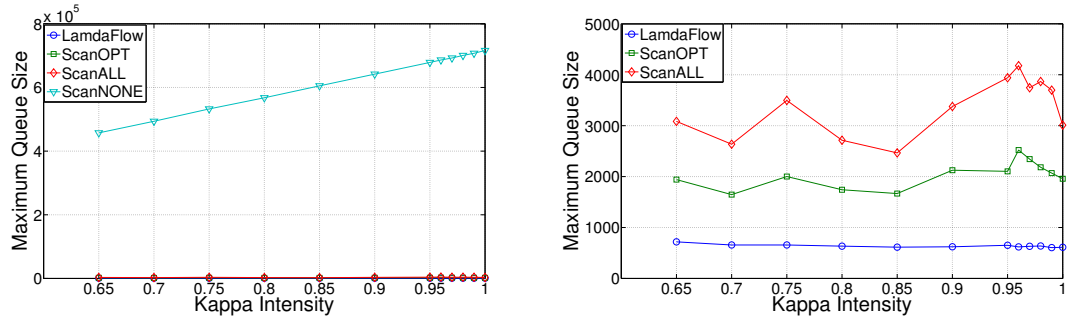
(a) (b) Zoomed version of (a)

Figure 5.12 Comparison of maximum latency of LambdaFlow, ScanOPT, ScanALL and ScanNONE strategies with varying Intensity of κ



(a) (b) Zoomed version of (a)

Figure 5.13 Comparison of average queue sizes of LambdaFlow, ScanOPT, ScanALL and ScanNONE strategies with varying Intensity of κ



(a) (b) Zoomed version of (a)

Figure 5.14 Comparison of maximum queue sizes of LambdaFlow, ScanOPT, ScanALL and ScanNONE strategies with varying Intensity of κ

(Hypothesis 4) where both the queues and latency of ScanNONE increase rapidly. ScanALL also performed worse than ScanOPT as in Hypothesis 5. The queues also were compared in Figures 5.13 and 5.14 where they show similar trend.

5.7 Summary

This chapter discussed centralized algorithms for load balancing in the cloud in the presence of malicious or unreliable packets. The model of the problem was described using standard models from the literature. SecureMaxWork, a centralized algorithm for load balancing in the cloud was introduced. Rigorous theoretical analyses of the algorithms was provided and it was shown that queues will stabilize. Extensive simulations proved that the ScanOPT strategy stabilized while the naive ScanALL and ScanNONE did not stabilize. Decentralized algorithms will be discussed in the next chapter.

Chapter 6

Reliable Job Scheduling in Cloud Computing - Decentralized Approaches

6.1 Overview

This chapter introduces robust decentralized algorithms for load balancing in the cloud. It is built from the the previous chapter which proposed centralized algorithms. The decentralized algorithms are introduced in Section 6.2 with some some theoretical justification. Extensive simulations and results are discussed in Section 6.3. Comparison between centralized and decentralized approaches is presented in Section 6.4, non-preemptive algorithms are discussed in Section 6.5 and conclusion is drawn in Section 6.6.

6.2 Decentralization

It was shown in the previous chapter that when the genuine job arrival rates λ_j and malicious jobs arrival rates κ_j, then if jobs are scanned with probability α_j queues will be stable and the mean latency will be bounded. Here the challenge is to propose several decentralized approaches to check if the queues and latency will the bounded.

Centralized approaches use the same queue for all type-j that are waiting in the system. In decentralized approach all servers maintain separate queues for jobs of type-j, therefore when a job arrives decision has to be made as to which server to route the job to. In this work 6 different algorithms are used.

Another distinct property of decentralized approach is the decision as to when to use the SecureMaxWork algorithm at each server to make scheduling decision. If preemptive algorithm is used then the problem will be trivial because SecureMaxWork will run at the

beginning of every time-slot at each server. Here however, non-preemptive scheduling are considered where jobs cannot be interrupted once they have started execution.

Maguluri et. al. [66] proposed the notion of *refresh time* A time slot t is a *global refresh time* if there is no job currently in service in all the servers at the beginning of t. *Local refresh time* happens when there is no job currently in service at the beginning of time slot t in a server i.

In practice global refresh times happen rarely and intuitively the happen to be more rear as the number of servers increase. Due to this phenomenon all the algorithms in this work will consider only local refresh times

Algorithm A: SecureMaxWork_JSQ Joint Shortest Queue (JSQ) paradigm is used to route a newly arrived job to the server with the queue with the smallest number of jobs of type-j, where j denotes the type of the arrived job. This algorithm was analyzed in the context of cloud workload in [66].

Algorithm B: SecureMaxWork_JSW. Joint Shortest Work (JSW) is used for routing a newly arrived job to the server with the minimum workload of type-j, where the workload is the sum of lengths of jobs in the local queue of type-j.

Algorithm C: SecureMaxWork_UR. Uniformly Random (UR) routing is used for forwarding newly arrived job to a server chosen uniformly at random.

Algorithm D: SecureMaxWork_RR. Round Robin (RR) routine is used for allocating newly arrived jobs to the servers, where RR is used separately for each type.

Algorithm E: SecureMaxWork_P2Q. Power-of-two-choices combined with selection of the Shortest Queue (P2Q) is used for routing a newly arrived job of a type-j: two servers are sampled uniformly at random, and the job is routed to the server with the shorter type-j queue.

Algorithm F: SecureMaxWork_P2W. Power-of-two-choices combined with selection of the Shortest Workload (P2W) is used for routing a newly arrived job of a type-j: two servers are sampled uniformly at random, and the job is routed to the server with the smaller workload of type-j (i.e., where the total length of type-j jobs in the local queue is shorter).

6.2.1 Stability of SecureMaxWork_JSW.

A rigorous mathematical analysis of a centralized scheduler SecureMaxWork with central queues was provided, where jobs could be distributed to various machines at any time after their arrival. Similar analysis applies to the decentralized SecureMaxWork with Join-Shortest-Work (JSW) routing policy. Recall that, in this policy, upon arrival of type-j job, sends it to machine that has minimum workload of type-j jobs, $Z_j^{(m)}$, which is defined in Section 5.2 but now computed for each machine separately. Then each machine tries to maximize work done,

$\max_{N \in S} \sum_{j=0}^{J} Z_j^{(m)}(t) N_j$. All the steps in the analysis of the main algorithm SecureMaxWork apply in this setting, resulting in almost identical analysis as in Section 5.4. An interesting open problem is to analyze mathematically the other five decentralized implementations of SecureMaxWork, and perhaps other similar decentralized algorithms.

6.3 Simulations

6.3.1 Experimental Setup

The experiment set up is the same as to that of Section 5.6, with virtual machines the same as Table 5.1 and the same servers. Job arrivals and job size distribution are also the same. Arrival vector of $\lambda = c \times [(1, 1/3, 2/3)]$ is used. The following optimal scanning vector α^* is computed for this setting, more precisely, the vector minimizing expected arriving weight:

- $\alpha^*_{i,1} = 0$ for $L_i \leq 2$,
- $\alpha^*_{i,2} = 0$ for $L_i \leq 34$,
- $\alpha^*_{i,3} = 0$ for $L_i \leq 50$,
- $\alpha^*_{i,j} = 1$ otherwise.

The only difference is that in this chapter the aim is to study how different routing protocols influence stability, when applied to the SecureMaxWork with the optimally selected scanning vector α^* above. The six decentralized implementations of the centralized algorithm, SecureMaxWork-OPT. They are denoted by ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_UR, ScanOPT_RR, ScanOPT_P2Q, and ScanOPT_P2W are compared. All the the measurements taken in Section 5.6 are also considered here and the results presented in the section below.

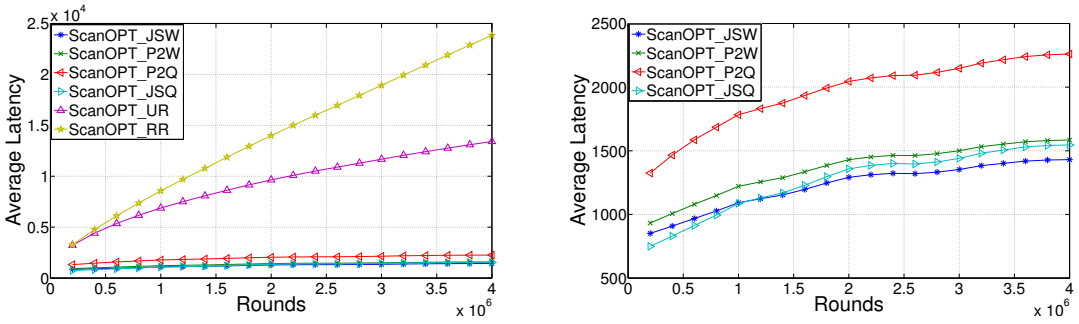
6.3.2 Results of Simulations

The aim of the experiments is to test the following hypotheses:

Hypothesis 6 *Based on the theory algorithms with the knowledge of the whole system are expected to be the best performing i.e. JSW and JSQ.*

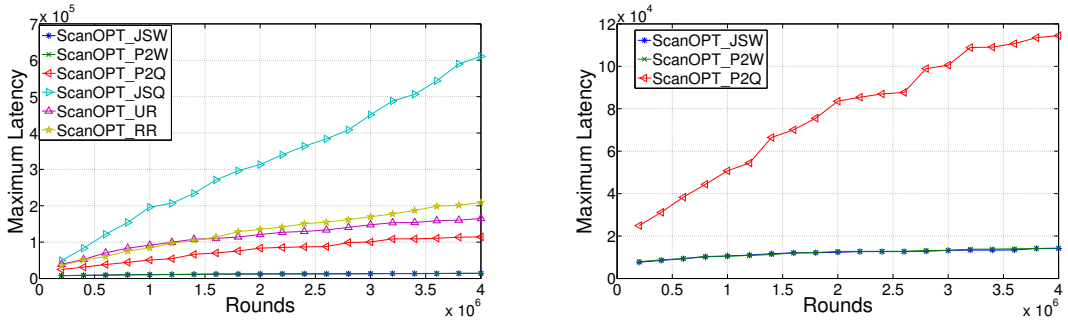
Hypothesis 7 *The algorithms based on power-of-two-choices should have better performance than the UR and RR because they consider some knowledge of the state of the servers as opposed to the purely random ones.*

Hypothesis 8 *The algorithms based on UR and RR are the worst performing because they do not have any smart strategy, in fact they will not be stabilizing.*



(a) (b) Zoomed version of (a)

Figure 6.1 Comparison of average latency using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR.



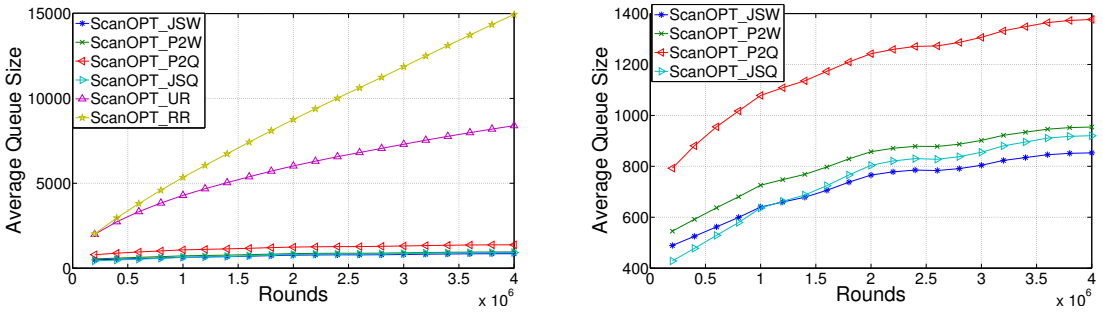
(a) (b) Zoomed version of (a)

Figure 6.2 Comparison of maximum latency using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR.

Figure 6.1 compares the latency of the six decentralized algorithms using ScanOPT strategy, i.e., ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR. The best performing algorithm is the one based on JSW (Hypothesis 6). This is followed by the one based on JSQ (Hypothesis 6), and then the two algorithms based on power-of-two-choices (Hypothesis 7). The worst performing algorithms are the ones based on round robin and uniform random selection, which grow rapidly (Hypothesis 8).

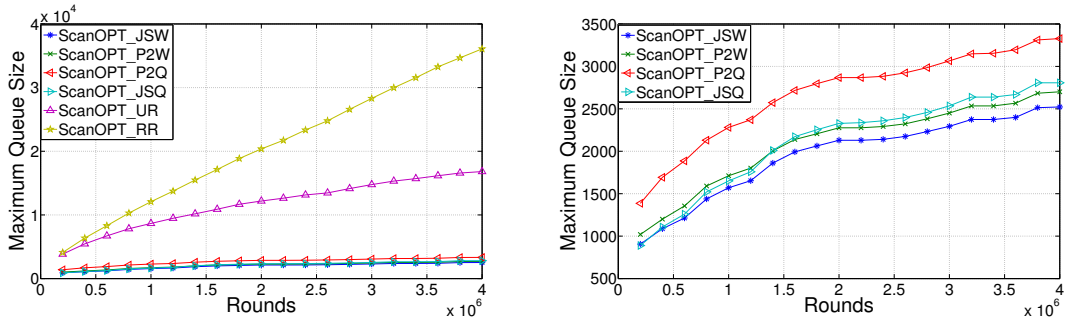
Figure 6.2, shows the trend for maximum latency. Where as expected, the algorithm based on JSW outperforms all the algorithms. A strange phenomenon noticed is that of JSQ. There is no clear explanation of this phenomenon, although it is suspected that this could be because choosing right configuration based on workload, as is done by SecureMaxWork,

causes long windows of time without feeding the local queues using the shortest workload policy could make these windows (and thus latency) even longer than using the shortest work paradigm.



(a) (b) Zoomed version of (a)

Figure 6.3 Comparison of average queue sizes using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR.



(a) (b) Zoomed version of (a)

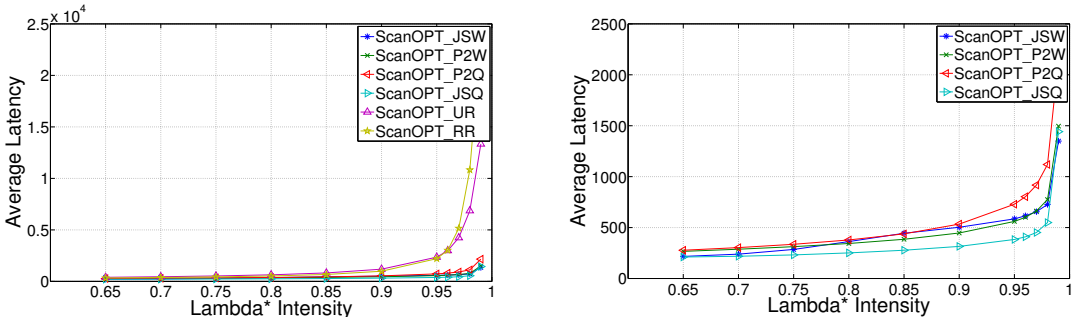
Figure 6.4 Comparison of maximum queue sizes using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR.

Figure 6.3 compares the average queue sizes of the 6 decentralized algorithms using ScanOPT strategy i.e. ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR. The best performing algorithm is the one based on JSW it is not surprising because it considers the exact amount of work left in all the servers to decide where to route new jobs. This is followed by JSQ and then the two algorithms based on power-of-two-choices. The worst performing algorithms are the ones based on round robin and uniform random selection which are not stabilizing. The performance is very similar for maximum queue size comparison shown in Figure 6.4.

Therefore the general trend is that the algorithms based on checking all the servers (JSQ and JSW)(Hypothesis 6) always outperform the ones based on power-of-two-choices (P2W

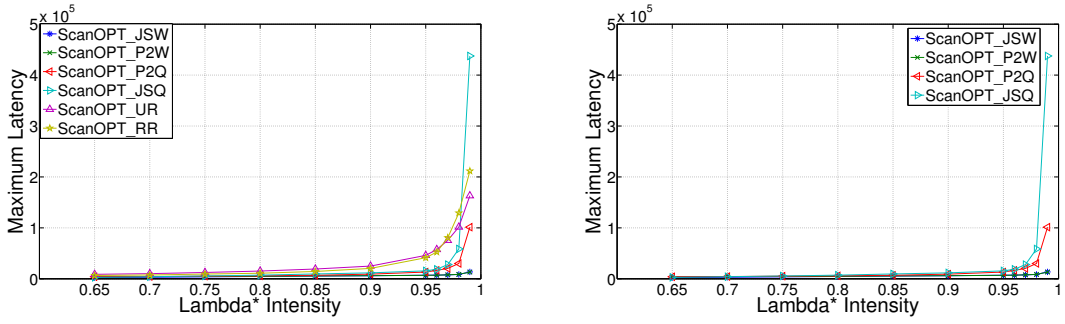
and P2Q), this phenomenon is not surprising because the algorithms based on all the servers consider the entire system state while the ones based on power-of-two-choices are random (Hypothesis 7). It should be noted that the algorithms based on power-of-two-choices are faster because decision can be made in constant time (which server to send an arriving job) while the ones based on all servers decision can only be made linear to the number of servers.

Hypothesis 9 *When λ is varied the result will be similar to Hypothesis 6, 7 and 8 where it is expected that JSW and JSQ will have the best performance followed by P2Q and P2W then the algorithms based on UR and RR.*



(a) (b) Zoomed version of (a)

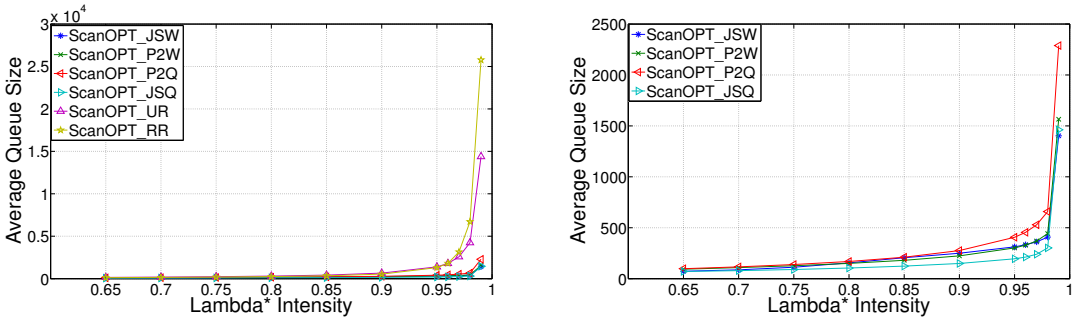
Figure 6.5 Comparison of average latency using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR with varying Intensity of λ



(a) (b) Zoomed version of (a)

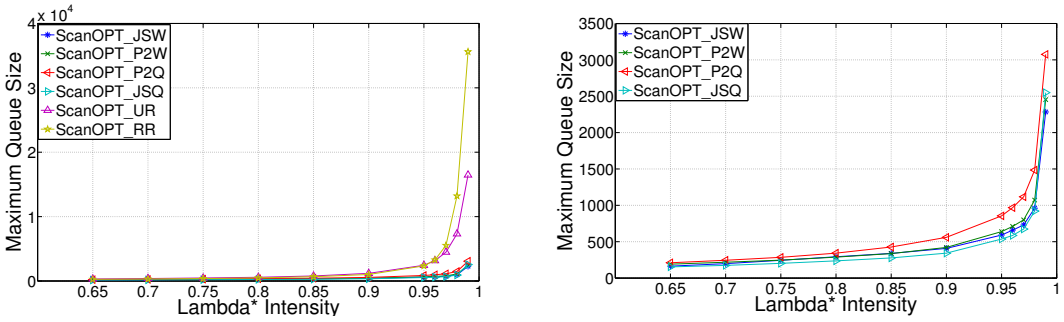
Figure 6.6 Comparison of maximum latency using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR with varying Intensity of λ

Figures 6.5, 6.6, 6.7 and 6.8 compares the average and maximum latency and queue of the six decentralized approaches varying λ intensities. The worst performing algorithms are



(a) (b) Zoomed version of (a)

Figure 6.7 Comparison of average queue sizes using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR with varying Intensity of λ



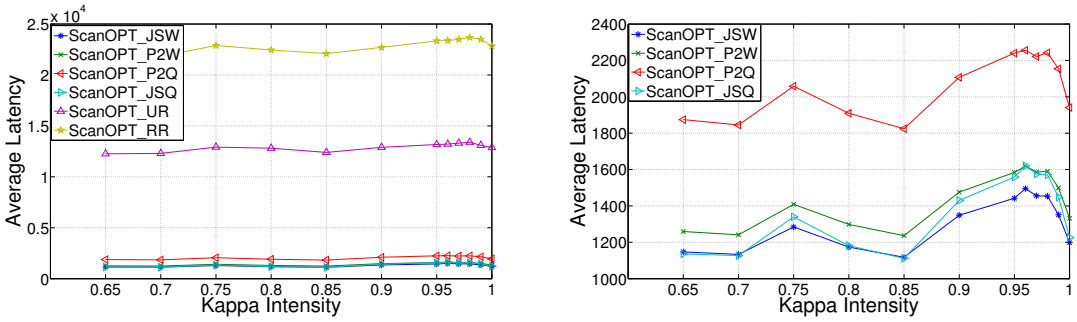
(a) (b) Zoomed version of (a)

Figure 6.8 Comparison of maximum queue sizes using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR with varying Intensity of λ

the ones based on RR and UR as they both explode from about intensity of $\lambda = 0.90$ for all measurements. For the average and maximum queue sizes and average latency performance of JSQ seems to be better initially but as the intensity increases the JSW outperforms the algorithm based on JSQ. The maximum latency remains high for the algorithm based on JSQ, the reason explained earlier all the figures support Hypothesis 9.

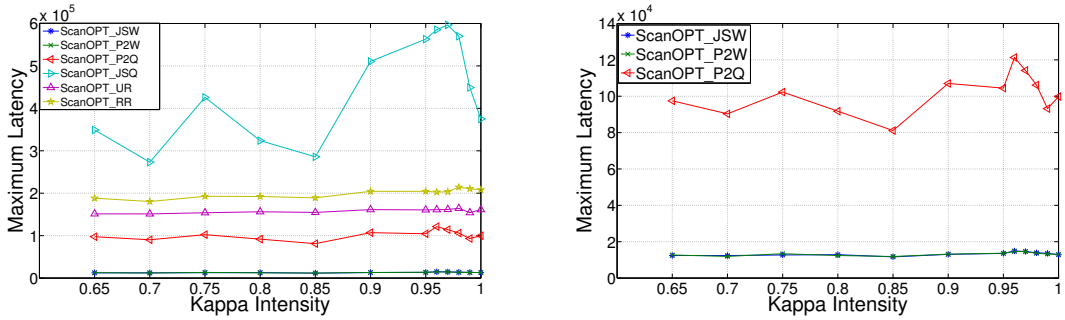
Hypothesis 10 *When κ is varied the result will be similar to Hypothesis 10 where it is expected that JSW and JSQ will have the best performance followed by P2Q and P2W then the algorithms based on UR and RR. There will be no burst because the proportion of κ in the traffic is rather small.*

Figure 6.9 shows the average latency of the decentralized approaches while varying κ intensity the algorithm based on JS* and P2* are almost 15 better than the one based on RR



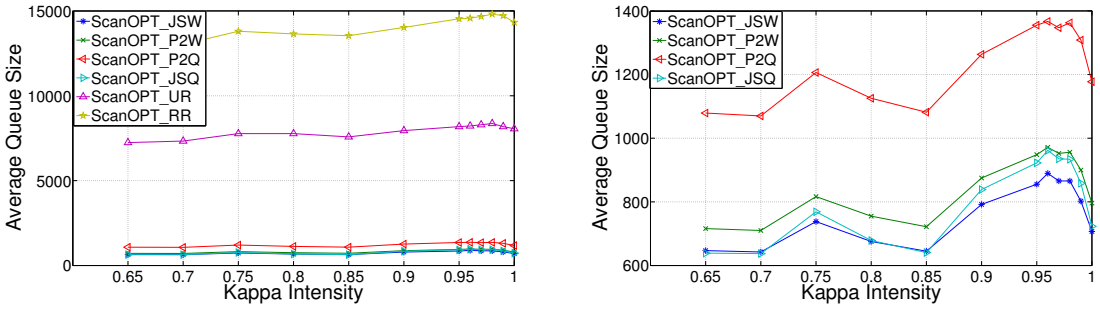
(a) (b) Zoomed version of (a)

Figure 6.9 Comparison of average latency using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR with varying Intensity of κ



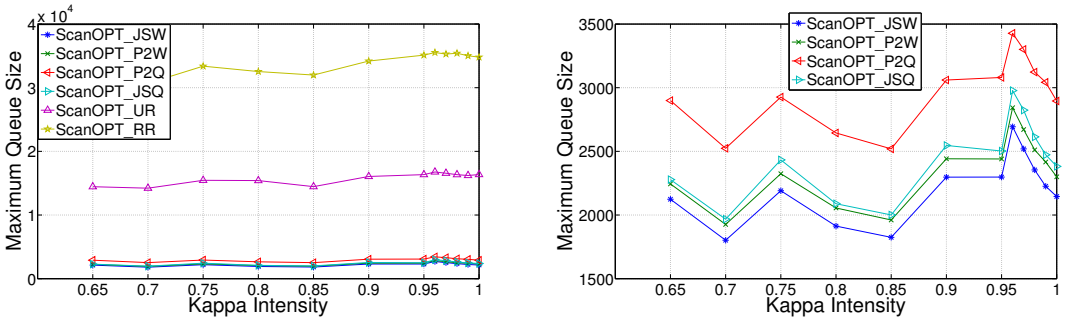
(a) (b) Zoomed version of (a)

Figure 6.10 Comparison of maximum latency using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR with varying Intensity of κ



(a) (b) Zoomed version of (a)

Figure 6.11 Comparison of average queue sizes using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR with varying Intensity of κ



(a) (b) Zoomed version of (a)

Figure 6.12 Comparison of maximum queue sizes using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR with varying Intensity of κ

and about 7 times better than the UR algorithm. This result is very similar to the one for maximum latency in Figure 6.10 and queue sizes in Figure 6.11, 6.12.

6.4 Comparison Between Centralized and Decentralized

Given that the experiment settings are identical between the centralized and decentralized approach, this section compares the performance of both the centralized and the decentralized algorithm. ScanNONE, ScanOPT_UR and ScanOPT_RR are not considered due to their poor performance.

Hypothesis 11 *The centralized approaches will perform better than the decentralized because all servers are utilized in the centralized case provided there are jobs waiting.*

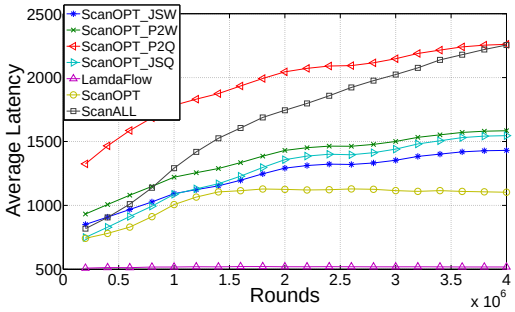


Figure 6.13 Comparison of average and maximum latency for decentralized and centralized approaches

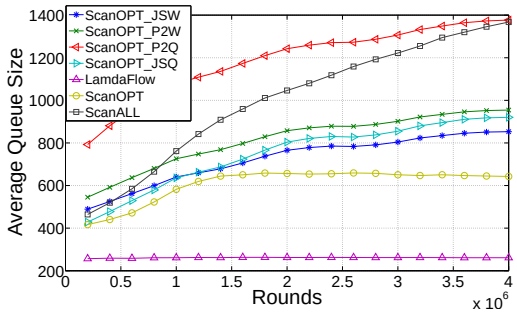


Figure 6.14 Comparison of average and maximum queue sizes for decentralized and centralized approaches

Figure 6.13 and 6.14 compares the latencies and queues sizes of centralized with decentralized approaches. The trend is that centralized ScanOPT performs better than all the decentralized algorithms for all the measurements. This supports Hypothesis 11.

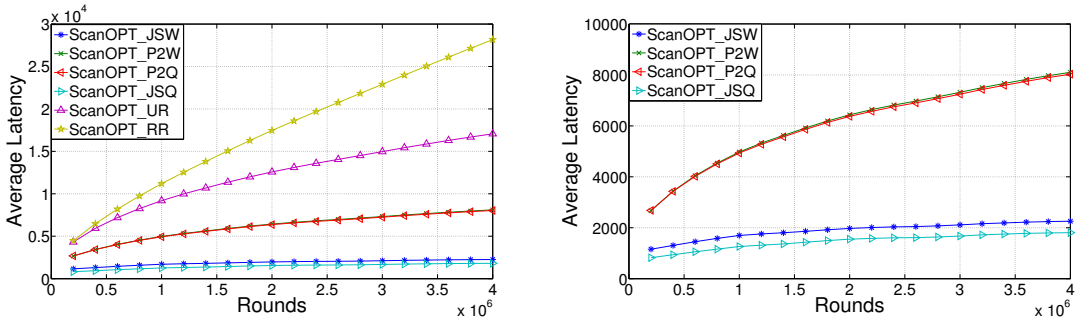
6.5 Non-preemptive Scheduling

This is the section where experiments are conducted for non-preemptive algorithms. The algorithms presented in the previous sections require that servers are configured and jobs are re-allocated at the beginning of each time slot. Some jobs may not be interrupt-able or interrupting a jobs could be costly (e.g the system needs to store a snapshot to be able to restart the VM later). Experiments are conducted using the same setting as in Section 6.3 but with *local refresh time* introduces in 6.2

Hypothesis 12 *Because Non-preemption is dependent on number of jobs waiting in the server, i.e. queue sizes, the algorithms based on shortest queue are expected to perform better.*

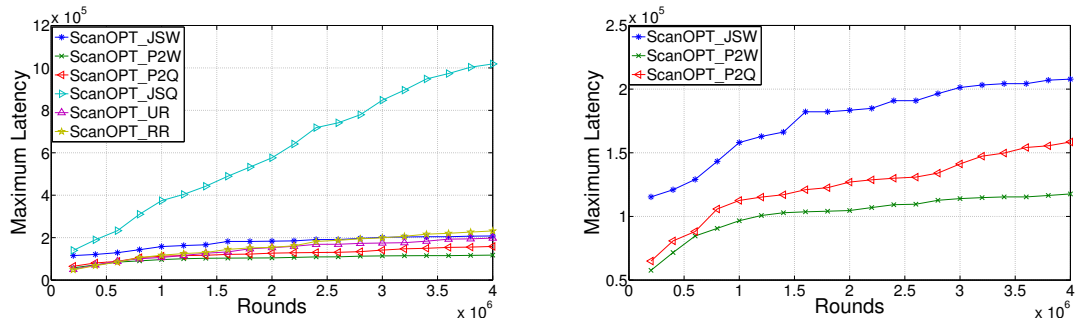
Figures 6.15 and 6.16 show the average and maximum latency of the 6 decentralized algorithms. From the previous experiments the algorithms based on RR and UR are the worst performing ones. The algorithms based on power-of-two-choices are very similar. But if observed more closely, the one based on queue performs slightly better than the one based on work (supporting Hypothesis 12). Finally the algorithms based on JS* are the best performing algorithms with average latency 4 times smaller than the power-of-two-choices counterparts. Proving Hypothesis 12 the algorithm based on JSQ slightly outperforms the one based on work.

The result for queue sizes is very similar to the results of latency. This is shown in Figure 7.1 and 6.18 further supporting Hypothesis 12.



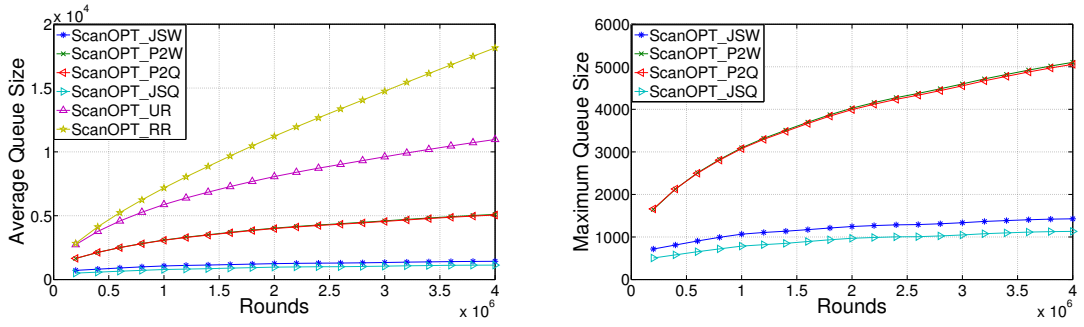
(a) (b) Zoomed version of (a)

Figure 6.15 Comparison of average latency for Non-preemptive ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR using local refresh times



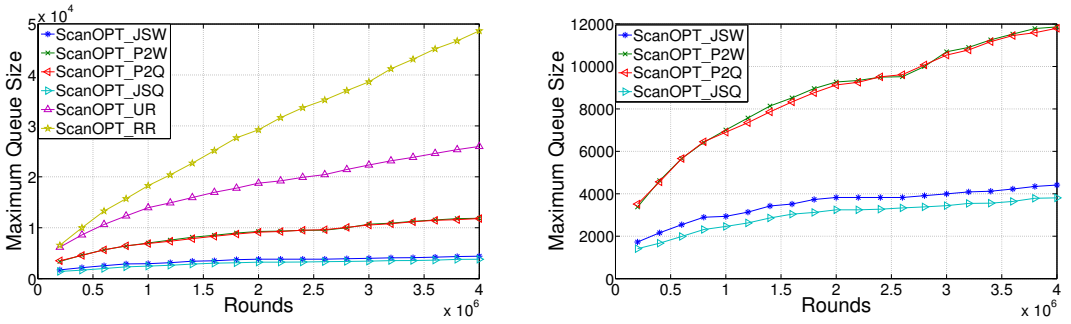
(a) (b) Zoomed version of (a)

Figure 6.16 Comparison of maximum latency for Non-preemptive ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR using local refresh times



(a) (b) Zoomed version of (a)

Figure 6.17 Comparison of average queue sizes for Non-preemptive ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR using local refresh times

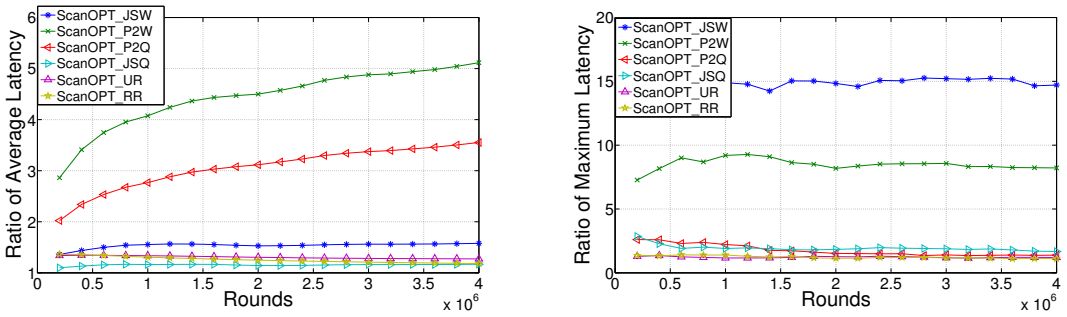


(a) (b) Zoomed version of (a)

Figure 6.18 Comparison of maximum queue sizes for Non-preemptive ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR using local refresh times

6.5.1 Comparison Between Pre-emptive and Non Pre-emptive

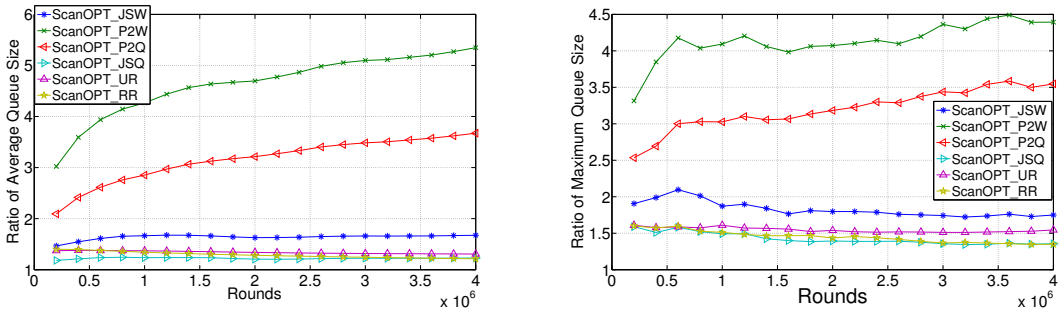
Hypothesis 13 *On algorithm by algorithm bases the preemptive algorithms are expected to be better than the non-preemptive ones because some resources will be wasted while jobs that have started are waiting to be finished.*



(a) (b)

Figure 6.19 Comparison of latencies of ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR

Figures 6.19 plots the ratio of latencies between non-preemptive and preemptive algorithms. All the algorithms have ratios greater than one, this suggests that the performance for non-preemptive is worse than for preemptive algorithms, this supports Hypothesis 13. The algorithm with the least ratio is the one based on JSQ followed by the algorithms based on RR and UR, all with ratio of average latency slightly about 1. These are closely followed by the algorithm based on JSW with ratio of above 1.5. The algorithms based on power-of-two-choices are the worse performing algorithms. P2Q has the ratio of about 3.5 and P2W has



(a) (b)

Figure 6.20 Comparison of queue sizes of ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR

ratio of about 5. There is no clear explanation to this phenomenon. The trend is very similar for the queue sizes in Figure 6.20.

6.6 Summary

Following the work done in Chapter 5 this chapter built on it and introduced decentralized scheduling for load balancing in cloud data canters. It was discussed that the centralized algorithm can be extended to the algorithm based on JSW. Experimental results proved the assertion where it perform better than all the algorithms discussed. It was closely followed by the algorithm based on JSQ, then the random algorithms based on power-of-two-choices. The algorithms based on UR and RR were the worst performing algorithms and they do not stabilize for the entire capacity region of the cloud. The chapter also compared the centralized and decentralized algorithms where it was observed as expected that the centralized algorithms performed better than the decentralized ones for both average queues and latencies. Finally, the chapter concluded by simulating non-preemptive algorithms, where it was discovered that the algorithms based on JSQ is the best performing algorithm because non-preemption considers queues sizes. Comparison of non-preemptive vs preemptive algorithms was also conducted where it was found that the non-preemption deteriorates performance of algorithms. This was not a surprise because resources are wasted when waiting for jobs to finish execution.

Chapter 7

Reliable Job Scheduling in Heterogeneous Cloud

7.1 Overview

This chapter considers heterogeneous settings where each server has different amount of resources available. This is because when cloud providers want to increase the capacity of their data centers, it is natural for them to upgrade to servers with bigger capacity. Consider the *distributed setting*, discussed in Chapter 6 in which each server has its own queues; upon arrival, a job request is forwarded to some server and stored in the server's local queue corresponding to the requested type of VMs. When the resources become available, the scheduling algorithm determines which set of jobs is to be served within the local queue in the server. The system is *stable* if the queues do not tend to increase without bound.

In Chapter 5, it was shown that for homogeneous servers in a data center there exists a stable algorithm given maximal arrivals rates of genuine and unreliable requests when proper scanning procedure is selected. This chapter aims to develop similar algorithms for a data center with heterogeneous servers. In addition to guarantee quality of service, job latency will also be measured, which is defined as the amount of time a job resides in the system since its arrival.

The reader is referred to Section 5.2 for the description of precise model and Section 5.3 for job scheduler SecureMaxWork. The discussion of the decentralized implementation of SecureMaxWork in Section 6.2. The rest of the chapter is organized as follows: the experimental setting and the evaluation results of the performance of these algorithms are presented in Section 7.2. Section 7.3, gives mathematical proofs that some popularly used

simple algorithms are not stable and further experiments were conducted to show their instability. Finally Section 7.4 provides conclusion for the chapter.

7.1.1 Features of the System

The extension of the study of managing workload in clouds under unreliable workload scenarios from homogeneous servers setting in Section 5.2 to heterogeneous servers setting. Extending the model in [15, 68], unreliable part of the traffic are detected by scanning only some specifically selected jobs without sacrificing too much resources.

- The theoretical model in Chapter 5 of capturing the essence of this conditional scanning is extended to the heterogeneous server setting.
- Several decentralized versions of the algorithm SecureMaxWork were proposed on the heterogeneous setting.
- Evaluation of the algorithms was done through extensive simulations, with respect to the maximum and average latency over time. The experiments illustrate that under a certain system capacity region and stochastic arrival pattern of genuine and unreliable jobs, coupling SecureMaxWork with server dispatching (routing) policies Shortest Queue First and Shortest Work First are stable while other routing policies power-of-two-choices, Round Robin and Uniform Random are unstable.
- Further support the experimental results by proving mathematically that power-of-two-choices, Round Robin and Uniform Random are unstable for a substantial amount of workloads even within the system capacity.

7.2 Simulation

7.2.1 Experiment Setting

Servers and VMs. Consider two types of servers in the data center, the first with the following configuration: 30 GB memory, 30 EC2 computing units and 4000 GB (4TB) storage space. Arriving jobs are served in the cloud based on three types of Virtual Machines described in Table 5.1. This gives three maximal configurations available at each server: $(2,0,0)$, $(1,0,1)$ and $(0,1,1)$. The second server has 68GB Memory, 80 EC2 computing units and 7168GB (7TB) of storage, based on the virtual machine configuration in Table **??**, this gives us maximal configuration $(4,0,0)$, $(2,2,0)$, $(0,0,4)$, $(3,1,0)$, $(3,0,1)$, $(1,3,0)$, $(0,3,2)$, $(1,0,3)$, $(0,2,3)$, $(2,1,1)$, $(2,0,2)$ and $(1,2,2)$.

Job arrivals. Use the generic arrival vector $\lambda^* = 0.99 \times (1, 1/3, 2/3)$ for genuine users' workload, which is located close to the border of server one capacity area taking all three maximal configurations and also close to the server two capacity area taking three of its maximal configurations $(4,0,0)$, $(2,2,0)$, $(0,0,4)$ (observe that λ^* is a normalized linear combination of the six configurations mentioned, additionally re-scaled by factor 0.99).

In each time step, a job of type $j = 1,2,3$ is selected with probability $\frac{\lambda_j^*}{130.5}$, and its length is chosen according to the length distribution described below with mean length 130.5 (calculation shown later).

Similarly as above, unreliable workload is defined using a generic arrival vector $\kappa^* = (0.7, 0.01, 0.01)$, and the procedure of generating an unreliable traffic is analogous as above for generating the genuine users' one. Note that each of the arrival rates λ^* and κ^* is within the capacity range of a server, whereas the combined work-flow rate $\lambda^* + \kappa^*$ is not.

Job size distribution.

When a new job is generated, with probability of 0.7 it is an integer uniformly distributed in the interval $[1,50]$, with probability of 0.15 it is an integer uniformly distributed in $[251,300]$, and with probability of 0.15 it is an integer uniformly distributed in $[451,500]$. Note that there are 150 possible job lengths, and the mean length is 130.5, as assumed in the definition of arrival rates.

Set up of simulations. The *homogeneous* setting used 100 servers of type one. In the *heterogeneous* setting consider 80 servers of type one and 10 of type two. Therefore the maximal feasible arrival rates in heterogeneous setting is:

- $80 \times \frac{1}{3} \times (3,1,2) = (80, 80/3, 160/3)$ for type one;
- $10 \times \frac{1}{3} \times (6,2,4) = (20, 20/3, 40/3)$ for type two

which in total is the same arrival rate as for the homogeneous setting $100 \times \frac{1}{3} \times (3,1,2) = (100, 100/3, 200/3)$. Based on these the overall arrival rates are: $\lambda = 100 \times \lambda^* = (99, 33, 66)$ for genuine workload, and $\kappa = 100 \times \kappa^* = (70, 1, 1)$ for unreliable workload – they are inside the capacity regions for both homogeneous and heterogeneous settings (note that the capacity region of homogeneous setting is slightly smaller than that of heterogeneous). The job size distribution is as specified above, same for each job type. The following optimal scanning vector α^* is computed for this setting, more precisely, the vector minimizing expected arriving weight:

- $\alpha_{i,1}^* = 0$ for $L_i \leq 2$,
- $\alpha_{i,2}^* = 0$ for $L_i \leq 34$,

- $\alpha^*_{i,3} = 0$ for $L_i \leq 50$,
- $\alpha^*_{i,j} = 1$ otherwise.

Each execution includes $4,000,000$ time steps. The *average latency* and *maximum latency* were computed at at every time step and the results were recorded at every $200,000$ steps. The experiments were ran 10 times and the averages of the results for each recorded time step was recorded.

In the first part, the results of the above measurements of SecureMaxWork applied on simultaneous genuine and malicious flows, with scanning defined by vector α^* (recall that it is ScanOPT). The aim is to study how different routing protocols influence stability, when applied to the SecureMaxWork with the optimally selected scanning vector α^*. The six decentralized implementations of SecureMaxWork are compared: namely, ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_UR, ScanOPT_RR, ScanOPT_P2Q, and ScanOPT_P2W.

In the second part of the simulations, arrival vector is varied by defining a parameter $\rho \in (0,1]$ called *traffic intensity*. Recall that the generic arrival vector $\lambda^* = 0.99 \times (1, 1/3, 2/3)$. Here, the arrival vector λ is varied as $\rho \times (1, 1/3, 2/3)$, and set $\rho \in \{0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 0.96, 0.97, 0.98, 0.99\}$. This means that the probabilities of the arrival of any of the three types of jobs reflect the different job intensities.

Each execution includes $4,000,000$ time steps. *Average and maximum* and *latency and queue size* is computed at every time step and values are recorded at the end of $4,000,000$ time steps. The experiments were ran 10 times and the averages of the results for each traffic intensity was recorded.

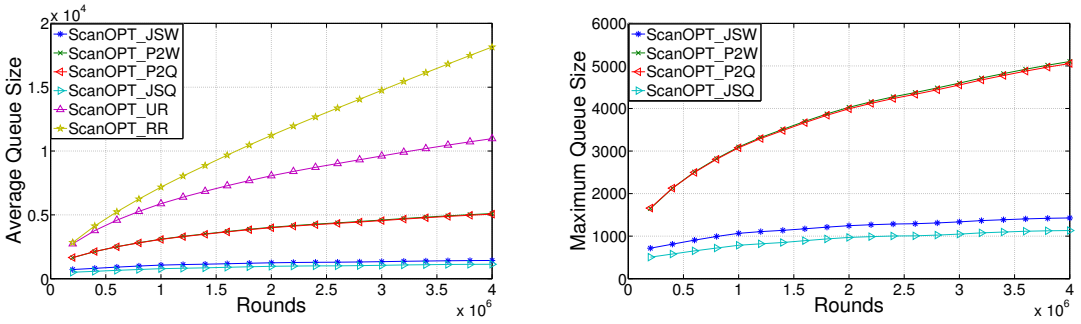
7.2.2 Results

Hypothesis 14 *Algorithms based on JSQ and JSW should stabilize with JSW performance better.*

Hypothesis 15 *Depending on arrival rates the algorithms based on P2Q and P2W should not stabilize.*

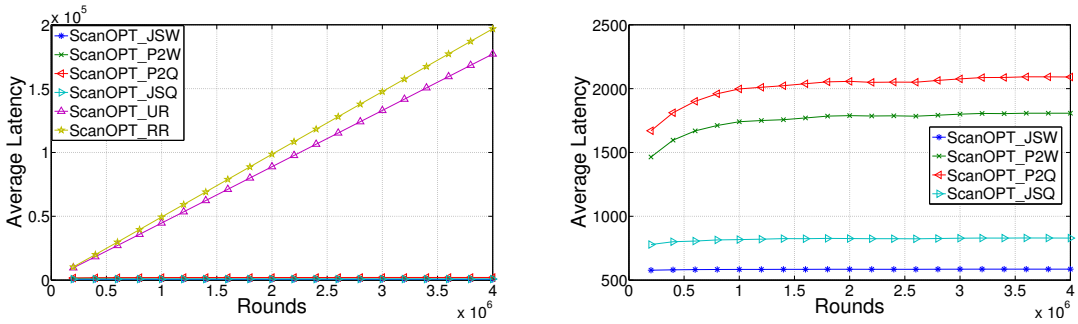
Hypothesis 16 *Algorithms based on RR an UR should not stabilized and should be substantially worse than the other four algorithms.*

Figure 7.2 compares average latency of the six decentralized algorithms using ScanOPT strategy. The best performing algorithms are the ones based on JSW and JSQ (Hypothesis14) followed by the two algorithms based on the power-of-two-choices P2W and P2Q (Hypothesis 15), In this case the algorithms seem to be stabilizing, this is because of the



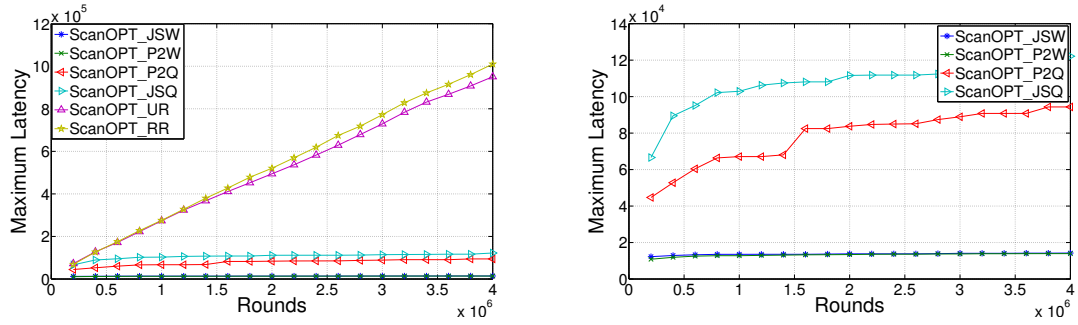
(a) (b) Zoomed version of (a)

Figure 7.1 Comparison of average queue sizes for Non-preemptive ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR using local refresh times



(a) (b) Zoomed version of (a)

Figure 7.2 Comparison of average latency using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR.



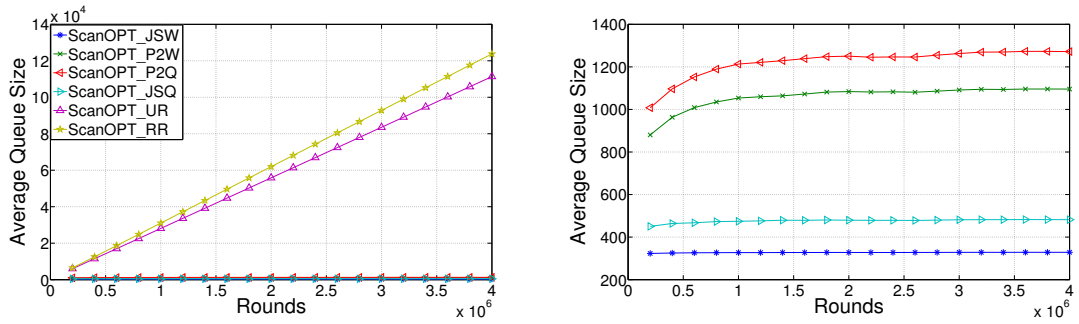
(a) (b) Zoomed version of (a)

Figure 7.3 Comparison of maximum latency using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR.

arrivals rates and the types of servers chosen. The worst performing algorithms are based on round robin and uniform random selection, which grow rapidly (Hypothesis 16). JSW and

JSQ (Hypothesis 14) are the best in terms of average latency performance, although the other two reasonable policies (P2W and P2Q) also show some stability trends. It was expected that the two algorithms based on power-of-two-choices (Hypothesis 15) will not perform as well as the JSQ and JSW (in fact they are 3 times worse) algorithms because they are random and majority of the selection will be from the small set of servers i.e. server one. Therefore, work will not be evenly distributed.

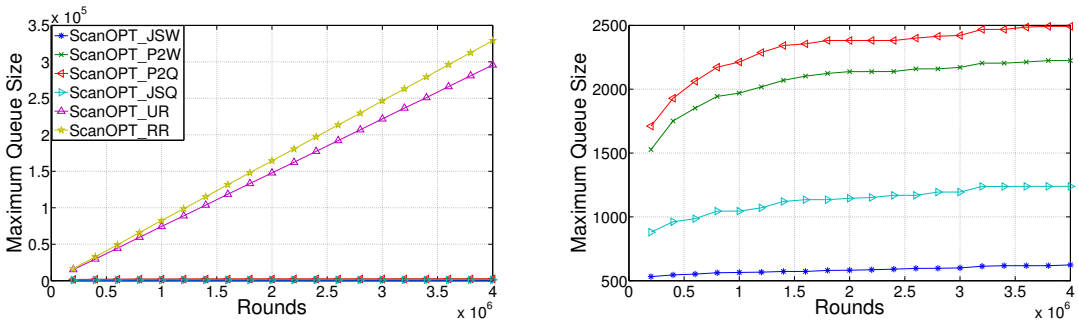
The result for maximum latency is very similar to that of average latency, only that the algorithm based on JSQ has very poor performance, this is illustrated in Figure 7.3. There is no clear explanation of this phenomenon, although it is suspected that this could be because choosing optimal configuration based on workload, as is done by SecureMaxWork, causes long windows of time without changing configuration, and feeding the local queues using the shortest workload policy could make these windows (and thus latencies) even longer than using the shortest queue paradigm. By definition, SecureMaxWork considers workload and if a long job is waiting in a server it might not be considered minimum based on JSQ and it will not contribute much in SecureMaxWork to be process. Similarly, the power-of-two-choices policy (i.e., controlled use of randomization) behaves better for shortest work than for shortest queue.



(a) (b) Zoomed version of (a)

Figure 7.4 Comparison of average maximum queue size using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR.

The figures for maximum and average queue sizes showed similar and expected result with the algorithms based on JSW performing best followed by the one based on JSQ. The algorithms based on power-of-two-choices seem to be stabilizing but this is because the chosen arrival rates of the other server to be exactly the same as the old server. These are arrival rates face the same face in the convex hull. In the next section it will be shown that if different arrivals rates are chosen (i.e. different face of the structure) even for small number of servers, these algorithms will not stabilize for the entire capacity region.

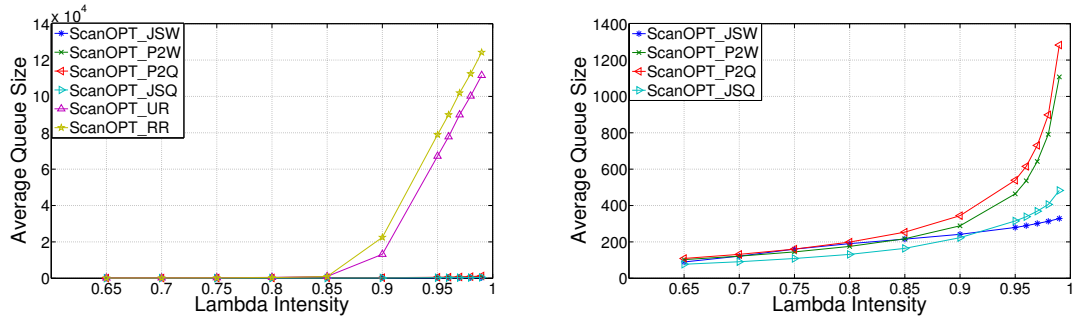


(a) (b) Zoomed version of (a)

Figure 7.5 Comparison of maximum queue size using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR.

Hypothesis 17 *The latency and queue sizes will be unstable at a lowest value of λ for UR and RR. .*

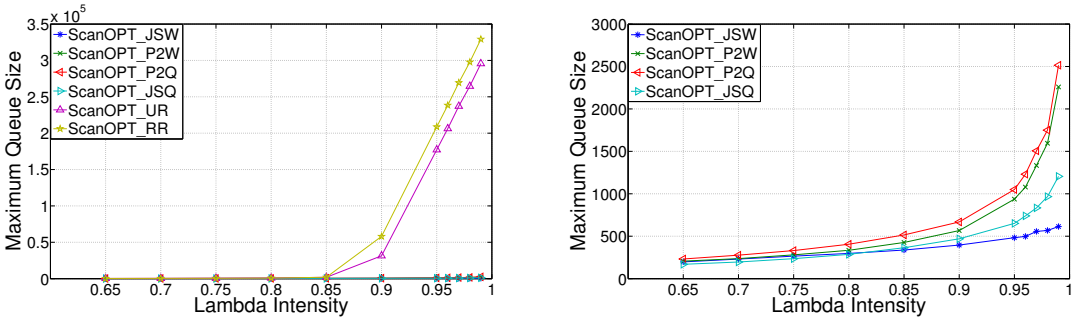
Hypothesis 18 *The latency and queue sizes will be unstable at a lower value of λ for P2* than for JS*.*



(a) (b) Zoomed version of (a)

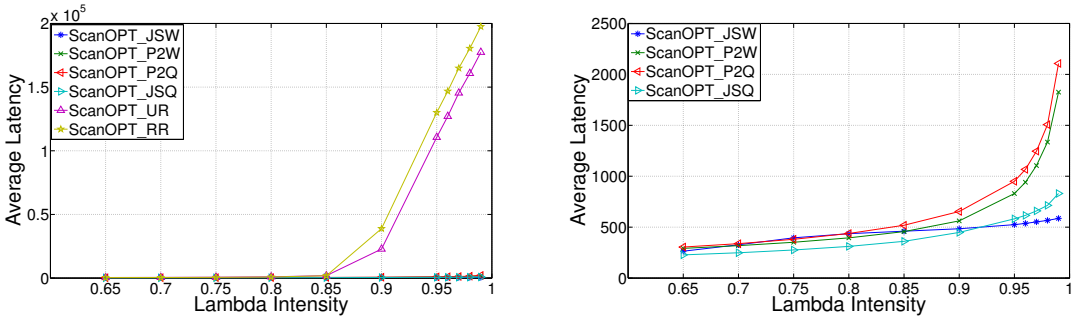
Figure 7.6 Comparison of average queue size for ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR using varying traffic intensities.

Figure 7.6 compares the average queue sizes of all algorithms using varying traffic intensities while in Figure 7.7 compares the maximum queue sizes. As expected, the algorithms based on uniform random and round robin allocation are not very stable because they lost stability as about $\lambda^* = 0.97$ (Hypothesis 17). The algorithms based on power-of-two-choices lose stability at around $\lambda^* = 0.97$ while the algorithms based on shortest queue and work are the most stable (Hypothesis 18). The trend is very similar in Figures 7.9 and 7.8 for maximum and average latency respectively.



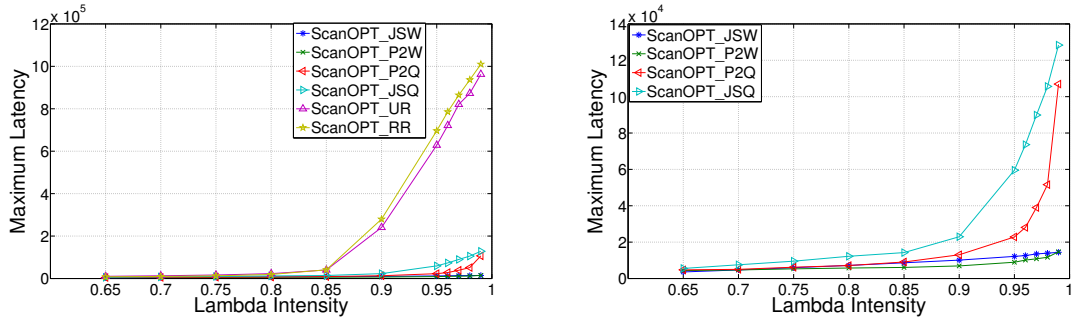
(a) (b) Zoomed version of (a)

Figure 7.7 Comparison of maximum queue size for ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR using varying traffic intensities.



(a) (b) Zoomed version of (a)

Figure 7.8 Comparison of average latency for ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR using varying traffic intensities.



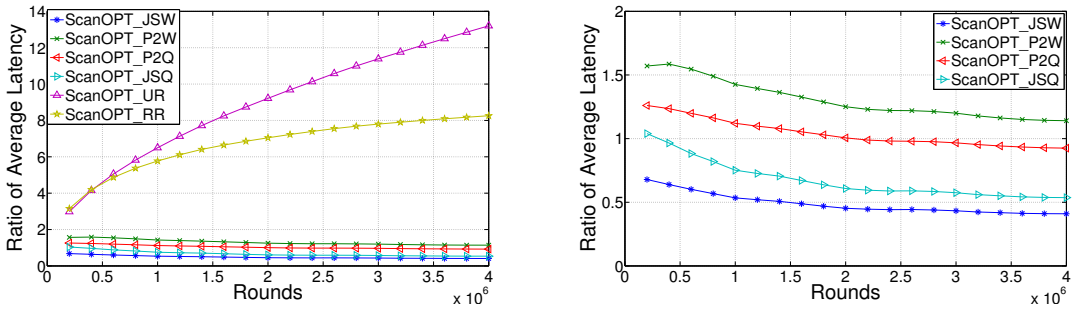
(a) (b) Zoomed version of (a)

Figure 7.9 Comparison of maximum latency for ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR using varying traffic intensities.

7.2.3 Comparison Between Heterogeneous and Homogeneous Data Centers

Given that we have the same arrival rates for the homogeneous simulation in Section 6.3.1 to Section 7.2.1 above two simulations where compared. Note that though the servers have the same total capacity. There are more available valid configurations for the 10 new bigger servers. The comparison is based on the ratio between all the parameters measure in heterogeneous setting to the ones measure in homogeneous setting.

It can be observed that the algorithms based on JSW and JSQ are doing better in this particular heterogeneous settings than in the homogeneous setting. This is because the bigger servers have more valid configurations but we are using similar arrival rates. The algorithms based on power-of-two-choices are behaving slightly worse in heterogeneous setting than the homogeneous setting. It is expected that the performance will be worse, the reason they are not worse is the same as the reason for good performance of JSW and JSQ approaches. The next section will show the expected results when different arrival rates are used for different servers.



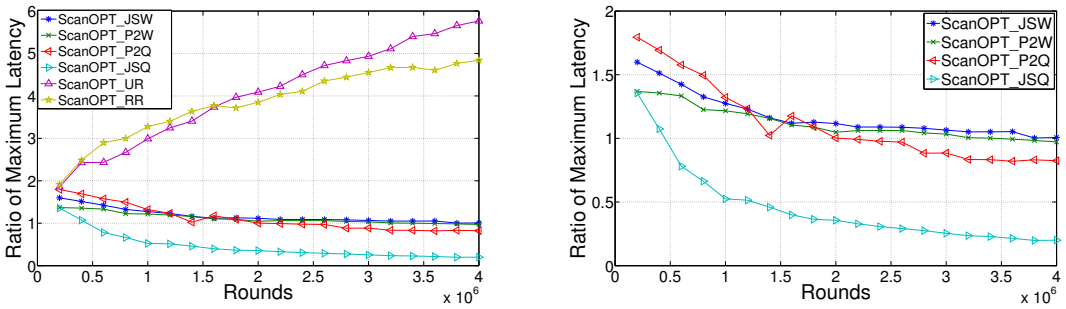
(a) (b) Zoomed version of (a)

Figure 7.10 Comparison of ratio of average latency between heterogeneous and homogeneous settings using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR.

7.3 Instability of Power of two Choices, Round Robin and Uniform Random policies

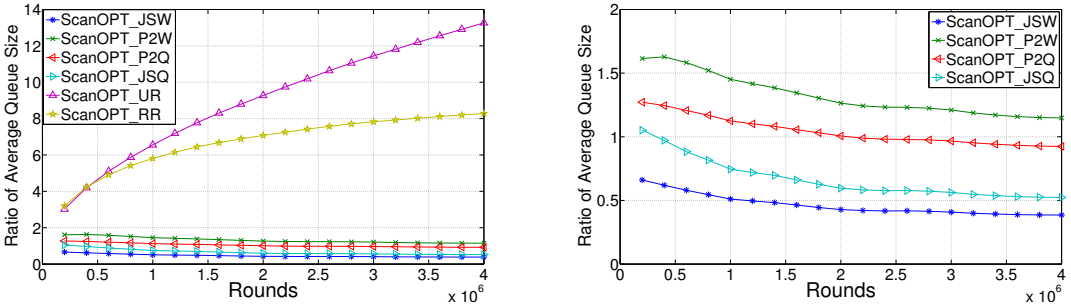
7.3.1 Introduction and Proofs

This section presents theoretical results that explain some of the results obtained from the experiments.



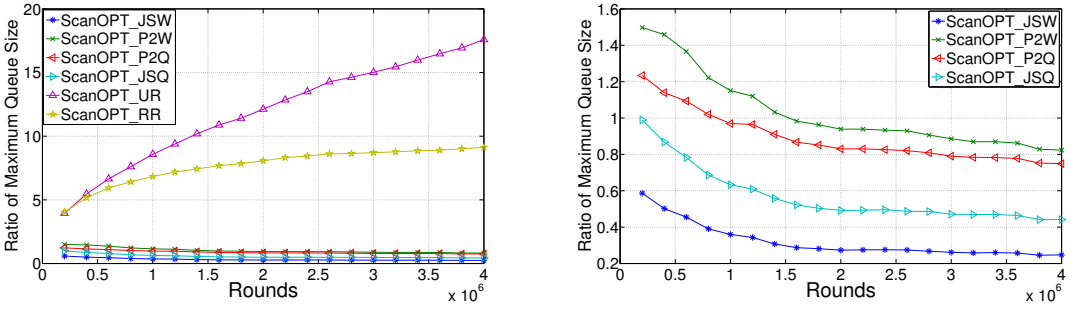
(a) (b) Zoomed version of (a)

Figure 7.11 Comparison of ratio of maximum latency between heterogeneous and homogeneous settingsusing ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR.



(a) (b) Zoomed version of (a)

Figure 7.12 Comparison of ratio of average queue size between heterogeneous and homogeneous settings using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR.



(a) (b) Zoomed version of (a)

Figure 7.13 Comparison of ratio of maximum queue size between heterogeneous and homogeneous settingse using ScanOPT_JSQ, ScanOPT_JSW, ScanOPT_P2Q, ScanOPT_P2W, ScanOPT_UR and ScanOPT_RR.

Theorem 5 *No job scheduler is stable for all arrival vectors inside the capacity region when combined with Power of two Choices routing policy.*

Proof: Consider the system of 80 small servers and 10 large servers as in Section 7.2.1. With probability $(80/90)^2$, Power of two Choices randomly picks two small servers and chooses one of them as destination for a considered job. This means that there are only 80 small servers that will receive at least $(8/9)^2$ of all the jobs.

Consider arrival rate of genuine jobs of $c \times [80 \times (1, 1/3, 2/3) + 10 \times (0, 2, 3)]$ for some $c < 1$. This arrival rate lies inside the capacity region of the system. Let x, y, z be the average number of small servers that chose configuration $(2,0,0)$, $(1,0,1)$ and $(0,1,1)$, respectively, per time slot. Note that for the system to be stable, the following must hold:

$$x + y + z = 80 \quad \text{(there are 80 small servers)}$$

and type-j jobs must be processed at least as frequently as they arrive:

$$\begin{aligned} 2x + y &\geq (8/9)^2 c \times (80 \times 1 + 10 \times 0) && \text{(type-1 jobs)} \\ z &\geq (8/9)^2 c \times (80 \times 1/3 + 10 \times 2) && \text{(type-2 jobs)} \\ y + z &\geq (8/9)^2 c \times (80 \times 2/3 + 10 \times 3) && \text{(type-3 jobs)} \end{aligned}$$

If the three inequalities are summed up:

$$2x + 2y + 2z \geq (8/9)^2 \times 210c$$

$$160 \geq (8/9)^2 \times 210c$$

$$81 \geq 4 \times 21c$$

Hence, for $c = 0,97$ no job scheduler can process jobs in small servers as fast as Power of two Choices injects them, despite the arrival rates being inside the capacity region.

■

Note that in the scenario described in the proof of Theorem 5, Round Robin and Uniform Random routing policies would direct $8/9$ jobs to the small servers instead of $(8/9)^2$, which makes the overload of the small servers even bigger than in case of Power of two Choices. Therefore, the conclusion is as follows.

Corollary 1 *No job scheduler is stable for all arrival vectors inside the capacity region when combined with Round Robin or Uniform Random routing policy.*

7.3.2 Experimental Setup

The experiment set up is very similar to that of Section 7.2. With the same VM and servers as in Table **??**. Arrival vector of $\lambda = c \times [(1,1/3,2/3)+(0,2,3)]$ is used. For $c = 0.97$ and $c = 0.96$. No malicious workload was considered in this case. The job size distribution remains the same as in Section 7.2 above. In this setting, 80 servers of type one and 10 of type two were used. Therefore the maximal feasible arrival rates in heterogeneous setting is:

- $80 \times \frac{1}{3} \times (3,1,2) = (80, 80/3, 160/3)$ for type one;
- $10 \times \frac{1}{3} \times (0,2,3) = (0, 20/3, 30/3)$ for type two

which gives us total arrival rate of $(80, 100/3, 190/3)$.

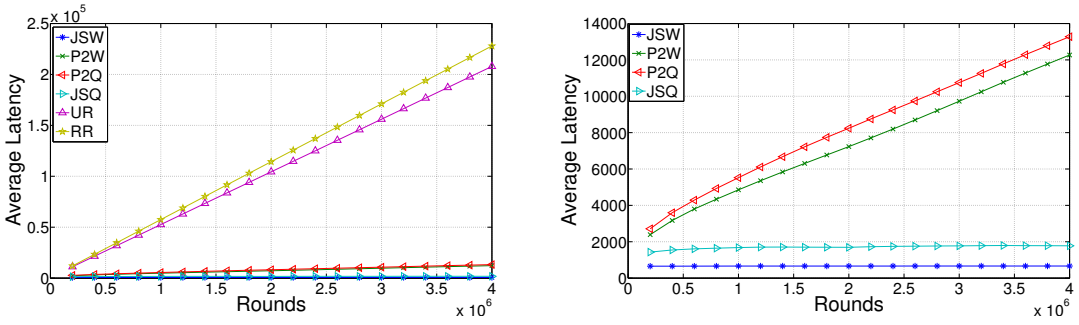
7.3.3 Results of Simulations

Instability of Power of two Choices for c = 0.97

The aim of the simulations is to confirm the conclusion that, for $c = 0,97$ no job scheduler can process jobs in small servers as fast as Power of two Choices injects them, despite the arrival rates being inside the capacity region. The following Hypotheses will be tested:

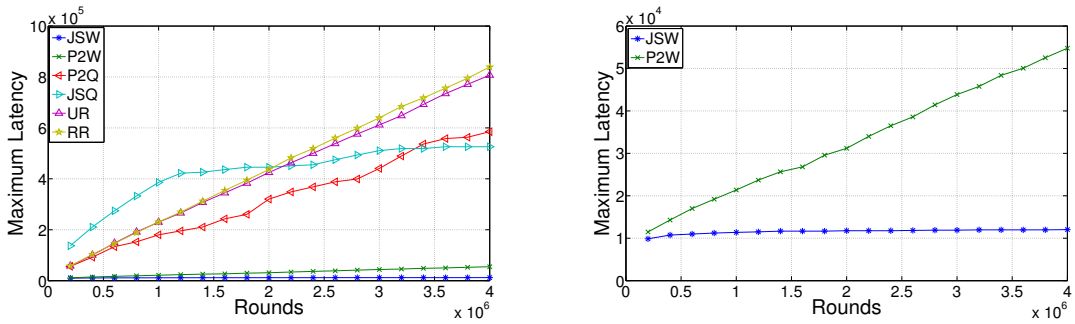
Hypothesis 19 *Based on the theory, algorithms based on power-of-two-choices, Round Robin and random routing are not stable for c=0.97.*

Hypothesis 20 *Based on the theory, algorithms based on joint shortest work/queue are stable*



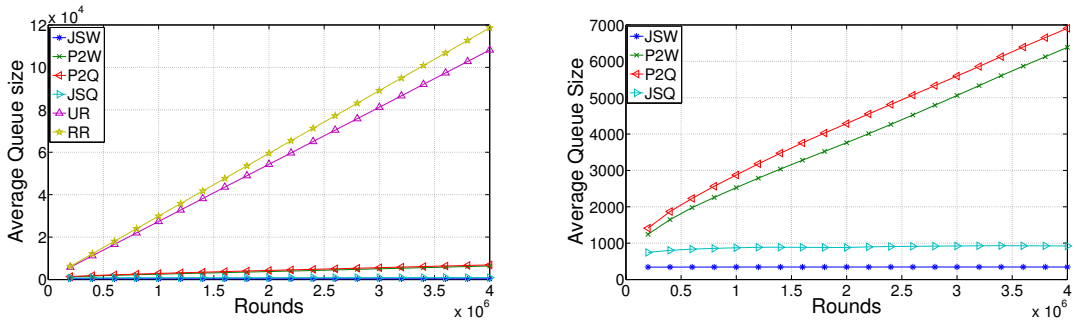
(a) (b) Zoomed version of (a)

Figure 7.14 Comparison of average latency using JSQ, JSW, P2Q, P2W, UR and RR for c = 0.97 using second setting



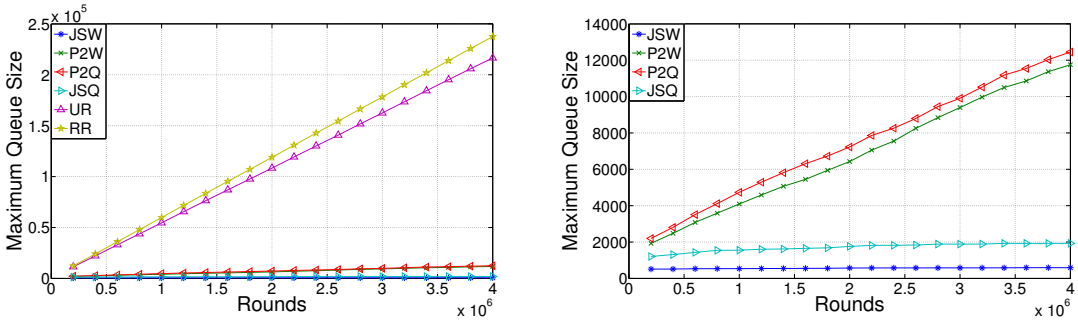
(a) (b) Zoomed version of (a)

Figure 7.15 Comparison of maximum latency using JSQ, JSW, P2Q, P2W, UR and RR for c = 0.97 using second setting



(a) (b) Zoomed version of (a)

Figure 7.16 Comparison of average queue sizes using JSQ, JSW, P2Q, P2W, UR and RR for c = 0.97 using second setting



(a) (b) Zoomed version of (a)

Figure 7.17 Comparison of maximum queue sizes using JSQ, JSW, P2Q, P2W, UR and RR for c = 0.97 using second setting

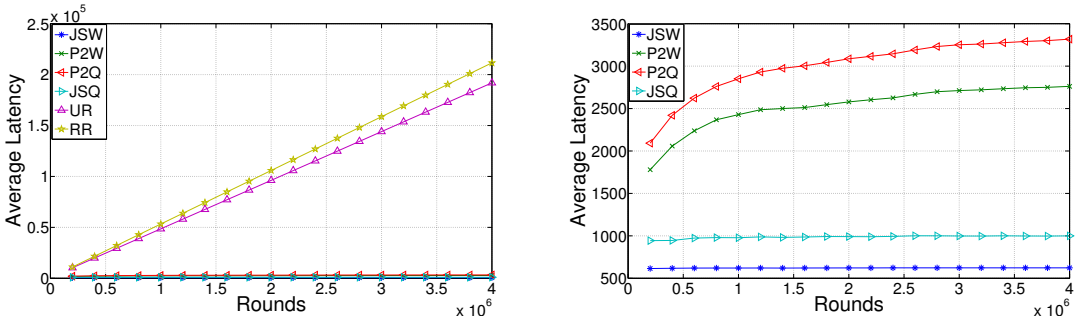
Figure 7.14 shows the average latency of the dispatching algorithms it is clear that the algorithms based on UR and RR are not stabilizing. If the figures are zoomed it can be clearly seen that the algorithms based on power-of-two-choices are also not stabilizing. The

algorithms based on JS* are stabilizing which supports Hypothesis 20. The figure is very similar for maximum latency in Figure 7.15 where all the algorithms based on random routing do not stabilize, supports Hypothesis 19.

Stability of Power-of-Two Choices for c = 0.96

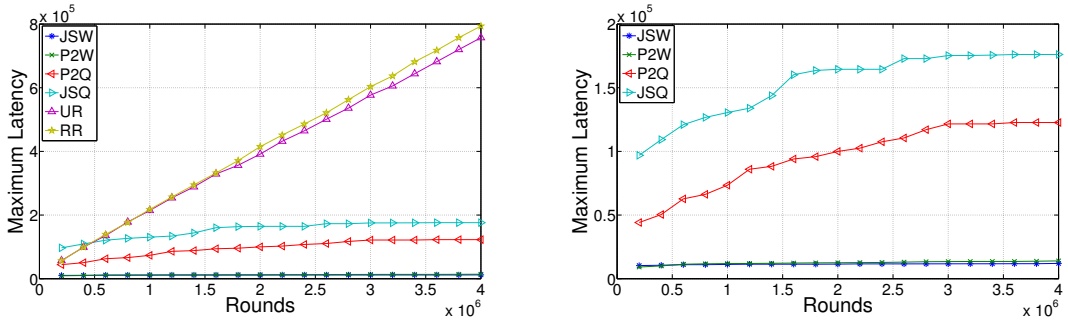
If the rates of $c = 0,96$ are used the algorithms based on Power of two Choices stabilizes. The following hypothesis will be tested.

Hypothesis 21 *Based on the theory, algorithms based on power-of-two-choices are stable for c=0.96.*



(a) (b) Zoomed version of (a)

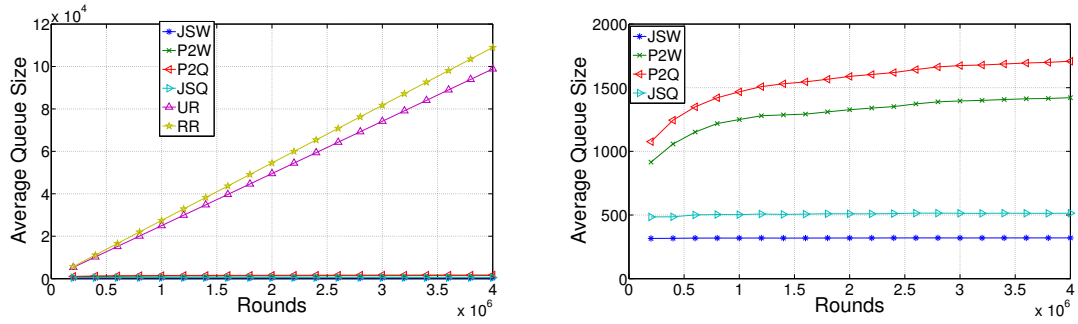
Figure 7.18 Comparison of average latency using JSQ, JSW, P2Q, P2W, UR and RR for c = 0.96 using second setting



(a) (b) Zoomed version of (a)

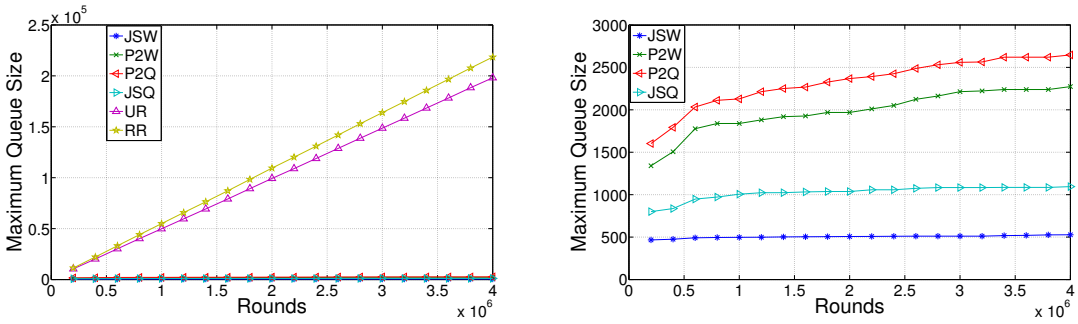
Figure 7.19 Comparison of maximum latency using JSQ, JSW, P2Q, P2W, UR and RR for c = 0.96 using second setting

Figure 7.18 shows the results of the simulation for average latencies while Figure 7.19 shows results maximum latencies. The algorithms based on JSQ and JSW are stabilizing



(a) (b) Zoomed version of (a)

Figure 7.20 Comparison of average queue sizes using JSQ, JSW, P2Q, P2W, UR and RR for c = 0.96 using second setting



(a) (b) Zoomed version of (a)

Figure 7.21 Comparison of maximum queue sizes using JSQ, JSW, P2Q, P2W, UR and RR for c = 0.96 using second setting

as expected but the aim of the experiments to test if the ones based on P2W and P2Q will stabilize. The figures indicate that both algorithms seem to be stable. This clearly supports the theory in Section 7.3.1 and supports Hypothesis 21. Note that the algorithms based on RR and UR are still not stable. The results for queue sizes are very similar to the ones above (see Figures 7.20 and 7.21) further supporting Hypothesis 21.

7.4 Summary

This chapter extends the work done in Chapter 6. The distributed algorithms for load balancing in the cloud were discussed. As data centers evolved, cloud providers upgrade their servers with bigger servers which makes the data centers heterogeneous. Theoretical analyses showed that the algorithms based on power-of-two-choices, uniform random and round robin are not stable for the entire cloud capacity. Experimental results confirmed the theoretical analysis.

Chapter 8

Conclusions and Future Work

8.1 Overview

This chapter provides summary of all the previous chapters, highlight main findings and results and proposes some directions for future research.

This chapter is organized as follows: Section 8.2 discusses the research contributions made in this thesis. These include: a hierarchical model for intrusion detection, auto scaling and cloud security, centralized reliable scheduling in cloud data centers and reliable job scheduling in heterogeneous cloud. Main findings are discussed in Section 8.3 and finally future work is proposed in Section 8.4.

8.2 Research Contribution

The main aim of this research is to rethink on how to look at security from the cloud perspective. With that it aims to propose a hierarchical model for intrusion detection in the cloud. Current research's aim is also to propose reliable scheduling algorithms for load balancing in the cloud.

8.2.1 Hierarchical Model for Intrusion Detection in the Cloud

To begin with a new model where different rules are checked at different levels of cloud computing hierarchy was introduced. Automata for all levels were produced and results show that the system stays in the desired state 60% of the time even if there is 10% of the traffic as attack.

8.2.2 Auto Scaling and Security in Cloud

This constitute a review of auto scaling and how it affects cloud computing clusters. Major commercial cloud computing providers were reviewed. A new model for handling auto scaling request was proposed, this model goes a little further to investigate why there is an auto-scaling request to protect cloud computing users. This research inspired the subsequent aims of looking at the relationship between load balancing and security in cloud computing data centers.

8.2.3 Centralized Reliable Scheduling in Cloud Data Centers

In this section SecureMaxWork was introduced, an algorithm that maximizes the total workload multiplied by all possible configurations while choosing some scanning strategy. It considers that scanning costs resources. It was shown theoretically that average queues will be stable even if arrival rates of combined malicious and genuine traffic are outside the cloud capacity region, provided the right scanning strategy is chosen. Implementation of the algorithm and experiments confirmed the theory, with the optimal scanning strategy stabilizing while the naive ScanAll strategy does not. Results showed that the latency of ScanALL can be more than twice that of ScanOPT, the difference in latency can be as much as 4000 and both of these parameters are growing. This trend is very similar for queue sizes. The ScanNONE strategy is increasing exponentially.

8.2.4 Decentralized Reliable Scheduling in Cloud Data Centres

The previous section uses central queues i.e. queues that are maintained by a central scheduler. In the decentralized approach when a job arrives, a decision is made as to which server to send a job and queues are maintained by server. The following six dispatching algorithms where analyzed using the same ScanOPT strategy:

Joint Shortest Work with SecureMaxWork

Considers the server with the least amount of work of a job type and sends the arriving job there. The best performing algorithm for all the performance measure. Two and half times better than UR and 5 times better than RR. The closest dispatching algorithm is JSQ with average queue difference of about 50 jobs and difference of 100 with P2W. Almost half the queue size of P2Q.

Joint Shortest Queue with SecureMaxWork

Considers the server with the least number of jobs of that type and sends the arriving job there. The second best performing algorithm for all the performance measure. The closest dispatching algorithm is JSW with average queue difference of about 50 jobs.

Power of Two Choices-Work with SecureMaxWork

When a job arrives two servers are chosen uniformly at random and the job is sent to the server with the least amount of work left of the two chosen server. This algorithm requires small amount of time and memory to make a decision. The performance is decent because it is just slightly worse than JSQ. For applications that require fast decisions this can be a good algorithm as well.

Power of Two Choices-Queue with SecureMaxWork

When a job arrives, two servers are chosen uniformly at random and the job sent to the server with the least number of jobs left of the two chosen server. Similar to the P2W this algorithm requires small amount of time and memory to make decision. The performance is decent because it is just slightly worse than JSQ. For applications that require fast decisions this can also be a good algorithm.

Uniform Random with SecureMaxWork

When a job arrives, a server is chosen uniformly at random and the job is sent to the server. It performs only better than the algorithm based on RR.

Round Robin with SecureMaxWork

When a job arrives, servers are chosen in a round robin manner. This is the worst performing algorithm and seems to be exploding for all the performance measures. This is because the state of a server is not considered before the jobs are sent to it.

Non-Preemptive Job Scheduling

The algorithms presented in the previous sections require that servers are reconfigured and jobs are re-allocated at the beginning of each time slot. This section considers a scenario where jobs are not interrupted once they start execution. Here the best performing algorithm was JSQ, this is because non-Preemption considers the amount of jobs waiting in the server. It is slightly better than the algorithm based on JSW. These algorithms are 4 times better than

the ones based on P2W and P2Q which are almost identical and have average queues of up to about 5000 jobs. As expected the algorithms based on RR and UR are the worst performing algorithms. Comparison between the preemptive with non preemptive showed that all the algorithms' performance for non-preemptive are worse. This is no surprise because resources will be wasted when waiting for jobs to finish before a new configuration is chosen.

8.2.5 Reliable Scheduling in Heterogeneous Cloud Data Centers

In the previous sections the homogeneous servers where considered but as cloud computing providers upgrade their data centers, they tend to upgrade with servers with higher capacity. Hence, it is inevitable that data centers become heterogeneous. Cloud computer data centers with heterogeneous were studied. Theoretical analysis showed that given some arrival rates the algorithms based on power-of-two-choices, RR and UR do not stabilize for the entire cloud capacity. For example two servers were used and it was shown that no stable algorithm exist for 97% of the capacity region (even without malicious traffic) for the four algorithms. Experimental results confirmed the theory and the 4 algorithms did not stabilize. When 96% was used however, the algorithms based on power-of-two-choices stabilized.

8.3 Main Results and Findings

- It was shown that the ScanOPT strategy stabilizes through formal analyses. Simulations proved this assertion even if the naive ScanALL will not be stable. It was also shown through simulations that the strategy ScanNONE is increasing at an exponential rate (Hypotheses 1 , 2 and 3).
- For decentralized scheduling, algorithms with the knowledge of the whole system were the best performing i.e. JSW and JSQ (Hypothesis 6).
- The algorithms based on power-of-two-choices have decent performance because although they are random, they consider some knowledge of the state of the servers (Hypothesis 7).
- The algorithms based on UR and RR are the worst performing because they do not have any smart strategy, in fact they will not be stable (Hypothesis 8).
- The centralized approaches perform substantially better than the decentralized because all servers are utilized in the centralized case provided there are jobs waiting (Hypothesis 11).

- Because Non-preemption is dependent on jobs waiting in the server, i.e. queue sizes, the algorithms based on shortest queue have substantially better performance (Hypothesis 12).
- On algorithm by algorithm bases the preemptive algorithms are expected to be few times better than the non-preemptive ones because some resources will be wasted while jobs that have started are waiting to be finished (Hypothesis 13).
- In heterogeneous systems there likely will be some ranges of arrival rates in which global queue knowledge algorithms will perform substantially better than those based on ad hoc local approach, and it was confirmed in sample scenario (Hypothesis 19 and 21).

8.4 Future Work

8.4.1 Hierarchical Model

The decentralized algorithms for load balancing introduce some hierarchy and potential for checking unreliability at different points in the cloud. For now scanning is only done at the server but decision to scan or not to scan is done centrally and uniformly for all servers. The model can be extended to different servers making different scanning decisions based on job types and utilization.

8.4.2 Non-preemptiveness

It was assumed that in regular time intervals all machines can be reconfigured — all jobs could be rescheduled and redistributed among the machines for further process. In some systems, interrupting execution of some jobs may be very costly. It is known how to transform such preemptive algorithms into ones with no reconfigurations, within (roughly) the same stability region, c.f., [15]. This technique applies also to our robust algorithms. One may consider a model, where a job, once started on a machine, can not be paused or rescheduled for completion in a different time, nor processed on a different machine.

The main idea in adapting SecureMaxWork to this model is to divide time into windows of length of T time slots. T should be large enough so that any job could be started at the start of the time window without breaking the above constraint. The algorithm will schedule jobs as previously with an additional constraint that only jobs that can be finished within a time window may be started. The stability analysis remains the same, except that the margin for unstable arrival rates should be made a bit larger to accommodate potential losses of

resources at the end of the time window. Thus, the higher T the better stability, though the latency may increase — studying this trade-off is an interesting open problem.

On the other hand, our results on non-stability presented in Theorem 5 and Corollary 1 automatically apply to the non-preemptive setting, as it is even more demanding (from perspective of job scheduler) than the preemptive setting analyzed in Theorem 5 and Corollary 1.

8.4.3 Algorithms without Knowledge of Arrival Rates

Finding the optimal scanning frequencies requires the knowledge of arrival rates of jobs of each type, length and genuine/malicious status. In practice, however, these values are not provided in advance. They can be estimated given a large enough sample.

If arrival rates are not provided, SecureMaxWork algorithm can be started using ScanALL strategy for a fixed but sufficiently long amount of time. During this time the status of genuine/malicious jobs can be learned (due to the ScanALL strategy) and therefore the algorithm will be able to estimate user-generated and malicious jobs arrival rates. The scanning frequencies that are optimal for the estimated arrival rates can be used. Note that using a different scanning strategy at the beginning for a fixed amount of time should not have impact on stability.

Furthermore, ScanALL strategy for a fixed amount of time can be used repeatedly, with significantly longer pause after each time (during which the scanning probabilities used will be computed based on the estimates), in order to enhance the quality of estimation of arrival rates, and thus using resources more and more efficiently. If pauses get long enough, this strategy should give better results than running ScanALL strategy only once.0

Another approach is to use ScanALL strategy once and then run the algorithm with optimal scanning frequencies for calculated estimations, but utilize information given by scanning jobs according to optimal scanning frequencies. Since job arrivals are i.i.d. among time slots, scanning x jobs randomly should give as good estimations as scanning first x jobs. Therefore, even when using optimal scanning frequencies, each time a job was scanned estimation of arrival rates improve. Designing and analyzing a stable algorithm for more scarce adversarial arrivals of malicious jobs is an interesting open problem.

8.4.4 Weighted Random and Round Robin in Heterogeneous Cloud

It was observed that some of the random algorithms that had decent performance for homogeneous cloud are not performing well in heterogeneous cloud. The reason might be because there is uniform random selection. It will be interesting to observe stability when weighted random algorithms are used. Weighing can be done based on arrival rates of different types

of jobs at servers and the number of those servers in the cloud system. This is especially true for the algorithms based on power-of-two-choices.

Bibliography

[1] Muhammed Abdulazeez, Pawel Garncarek, and Prudence W.H. Wong. Lightweight framework for reliable job scheduling in heterogeneous clouds. In *Proceedings of the 26th International Conference on Computer Communication and Networks (ICCCN-IoTPST)*, pages 1–6, 2017.

[2] Muhammed Abdulazeez, Dariusz R. Kowalski, Pawel Garncarek, and Prudence W.H. Wong. Lightweight robust framework for workload scheduling in clouds. In *Proceedings of the 1st International Conference on Edge Computing (EDGE)*, pages 1–4, 2017.

[3] Muhammed Bello Abdulazeez, Dariusz R Kowalski, Alexei Lisista, and Sultan S Alshamrani. Failure or denial of service? a rethink of the cloud recovery model. In *Proceedings of the 15th European Conference on Cyber Warfare and Security*, page 1, 2016.

[4] Muhammed Abdulazeez and Dariusz R Kowalski. Hierarchical model for intrusion detection systems in the cloud environment. In *Proceedings of the 14th European Conference on Cyber Warfare and Security*, page 319, 2015.

[5] Gaurav Somani, Manoj Singh Gaur, Dheeraj Sanghi, Mauro Conti, and Rajkumar Buyya. {DDoS} attacks in cloud computing: Issues, taxonomy, and future directions. *Computer Communications*, 107:30 – 48, 2017.

[6] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *2008 Grid Computing Environments Workshop*, pages 1–10, 2008.

[7] Petr Mell and Timothy Grance. The NIST definition of cloud computing. *National Institute of Standards and Technology*, 2011.

[8] Ahmed Patel, Mona Taghavi, Kaveh Bakhtiyari, and Joaquim Celestino JúNior. An intrusion detection and prevention system in cloud computing: A systematic review. *Journal of network and computer applications*, 36(1):25–41, 2013.

[9] Aaron Weiss. Computing in the clouds. *Computing*, 16, 2007.

[10] Michael Armbrust et al. Above the clouds: A berkeley view of cloud computing. 2009.

[11] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, and Dawn Leaf. Nist cloud computing reference architecture. *NIST special publication*, 500(2011):292, 2011.

[12] Subashini Subashini and V Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications*, 34(1):1–11, 2011.

[13] Ashley Chonka, Yang Xiang, Wanlei Zhou, and Alessio Bonti. Cloud security defence to protect cloud computing against http-dos and xml-dos attacks. *Journal of Network and Computer Applications*, 34(4):1097–1107, 2011.

[14] Chirag Modi, Dhiren R. Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttukrishnan Rajarajan. A survey of intrusion detection techniques in cloud. *Journal Network and Computer Applications*, 36(1):42–57, 2013.

[15] Siva Theja Maguluri, R. Srikant, and Lei Ying. Stochastic models of load balancing and scheduling in cloud computing clusters. In *Proceedings of the of International Conference on Computer Communications (INFOCOM)*, pages 702–710, 2012.

[16] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism: Probabilistic symbolic model checker. In *Proceedings of the International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*, pages 200–204, 2002.

[17] Shane M Greenstein and James B Wade. The product life cycle in the commercial mainframe computer market, 1968-1982. *The RAND Journal of Economics*, pages 772–789, 1998.

[18] Gerald J Popek and Robert P Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, 1974.

[19] Paul Ferguson and Geoff Huston. What is a vpn?, 1998.

[20] Marios D Dikaiakos, Dimitrios Katsaros, Pankaj Mehra, George Pallis, and Athena Vakali. Cloud computing: Distributed internet computing for it and scientific research. *IEEE Internet computing*, 13(5), 2009.

[21] Antonio Regalado. Who coined 'cloud computing'?, Dec 2013.

[22] Subashini Subashini and Veeraruna Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of network and computer applications*, 34(1):1–11, 2011.

[23] Yashpalsinh Jadeja and Kirit Modi. Cloud computing-concepts, architecture and challenges. In *Proceedings of the International Conference on Computing, Electronics and Electrical Technologies (ICCEET)*, pages 877–880, 2012.

[24] George Pallis. Cloud computing: the new frontier of internet computing. *IEEE internet computing*, 14(5):70–73, 2010.

[25] Kuyoro So. Cloud computing security issues and challenges. *International Journal of Computer Networks*, 3(5):247–55, 2011.

[26] Mariana Carroll, Alta Van Der Merwe, and Paula Kotze. Secure cloud computing: Benefits, risks and controls. In *Proceedings of the 10th International Information Security South Africa Conference(ISSA)*, pages 1–9, 2011.

[27] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: issues and challenges. In *Proceedings of the 24th International Conference on Advanced Information Networking and Applications (AINA)*, pages 27–33, 2010.

[28] Wei-Tek Tsai, Xin Sun, and Janaka Balasooriya. Service-oriented cloud computing architecture. In *Proceedings of the 7th International Conference on Information Technology: New Generations (ITNG)*, pages 684–689, 2010.

[29] Won Kim, Soo Dong Kim, Eunseok Lee, and Sungyoung Lee. Adoption issues for cloud computing. In *Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia*, pages 2–5, 2009.

[30] Won Kim. Cloud computing: Today and tomorrow. *Journal of object technology*, 8(1):65–72, 2009.

[31] Hassan Takabi, James BD Joshi, and Gail-Joon Ahn. Security and privacy challenges in cloud computing environments. *IEEE Security & Privacy*, 8(6):24–31, 2010.

[32] Dimitrios Zissis and Dimitrios Lekkas. Addressing cloud computing security issues. *Future Generation computer systems*, 28(3):583–592, 2012.

[33] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[34] Patrick Nelson. Cybercriminals moving into cloud big time, report says. `http://www.networkworld.com/article/2900125/malware-cybercrime/criminals-moving-into-cloud-big-time-says-report.html`, 2015. Accessed 13th January, 2017.

[35] Bruce Schneier. We still don't know who hacked sony. `https://www.theatlantic.com/international/archive/2015/01/we-still-dont-know-who-hacked-sony-north-korea/384198/`, 2015. Accessed 9th October, 2015.

[36] Cloud Tech. Cyber criminals compromising virtual machines in cloud to increase scale of ddos. `https://www.cloudcomputing-news.net/news/2016/dec/16/cyber-criminals-compromising-virtual-machines-cloud-increase-scale-ddos/`, 2016. Accessed 13th January, 2017.

[37] Liam Tung. Attackers probe google cloud, aws, microsoft azure customers for weaknesses. `http://www.cso.com.au/article/609597/attackers-probe-google-cloud-aws-microsoft-azure-customers-weaknesses/`, 2016. Accessed 13th January, 2017.

[38] Cade Metz. Ddos attack rains down on amazon cloud. `https://www.theregister.co.uk/2009/10/05/amazon_bitbucket_outage/`, 2009. Accessed 9th October, 2015.

[39] Berkeley Lovelace Jr. and Antonio José Vielma. Friday's third cyberattack on dyn 'has been resolved,' company says. `http://www.cnbc.com/2016/10/21/major-websites-across-east-coast-knocked-out-in-apparent-ddos-attack.html`, 2016. Accessed 13th January, 2017.

[40] Ddos-ers launch attacks from amazon ec2. `https://www.infosecurity-magazine.com/news/ddos-ers-launch-attacks-from-amazon-ec2/`, 2014. Accessed 9th October, 2015.

[41] Michael E Whitman and Herbert J Mattord. *Principles of information security*. Cengage Learning, 2011.

[42] Jerome H Saltzer and Michael D Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[43] Jenshiub Liu, Zhi-Jian Lee, and Yeh-Ching Chung. Efficient dynamic probabilistic packet marking for ip traceback. In *Proceedings of the 11th International Conference on Networks (ICON)*, pages 475–480, 2003.

[44] Jelena Mirkovic and Peter Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[45] Jie Yu, Zhoujun Li, Huowang Chen, and Xiaoming Chen. A detection and offense mechanism to defend against application layer ddos attacks. In *Proceedings of the 3rd International Conference on Networking and Services (ICNS)*, pages 54–54, 2007.

[46] Yi Xie and Shun-Zheng Yu. Monitoring the application-layer ddos attacks for popular websites. *IEEE/ACM Transactions on Networking (TON)*, 17(1):15–25, 2009.

[47] Hervé Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822, 1999.

[48] Leonid Portnoy, Eleazar Eskin, and Sal Stolfo. Intrusion detection with unlabeled data using clustering. In *Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA)*. Citeseer, 2001.

[49] Karen Scarfone and Peter Mell. Guide to intrusion detection and prevention systems (idps). *NIST special publication*, 800(2007):94, 2007.

[50] Hervé Debar, Marc Dacier, and Andreas Wespi. A revised taxonomy for intrusion-detection systems. *Annals of Telecommunications*, 55(7):361–378, 2000.

[51] Christopher Kruegel and Thomas Toth. Using decision trees to improve signature-based intrusion detection. In *Proceedings of the International Workshop on Recent Advances in Intrusion Detection*, pages 173–191. Springer, 2003.

[52] Theuns Verwoerd and Ray Hunt. Intrusion detection techniques and approaches. *Computer communications*, 25(15):1356–1365, 2002.

[53] Thomas Vissers, Thamarai Selvi Somasundaram, Luc Pieters, Kannan Govindarajan, and Peter Hellinckx. Ddos defense system for web services in a cloud environment. *Future Generation Computer Systems*, 37:37–45, 2014.

[54] Ashley Chonka, Wanlei Zhou, Jaipal Singh, and Yang Xiang. Detecting and tracing ddos attacks by intelligent decision prototype. In *Proceedings of the 6th Annual International Conference on Pervasive Computing and Communications (PerCom)*, pages 578–583, 2008.

[55] Dawei Wang, Longtao He, Yibo Xue, and Yingfei Dong. Exploiting artificial immune systems to detect unknown dos attacks in real-time. In *Proceedings of the 2nd International Conference on Cloud Computing and Intelligent Systems (CCIS)*, pages 646–650, 2012.

[56] Yiqiu Fang, Fei Wang, and Junwei Ge. A task scheduling algorithm based on load balancing in cloud computing. In *Proceedings of the International Conference on Web Information Systems and Mining*, pages 271–277, 2010.

[57] Zhuoyao Wang, Majeed M Hayat, Nasir Ghani, and Khaled B Shaban. A probabilistic multi-tenant model for virtual machine mapping in cloud systems. In *Proceedings of the 3rd International Conference on Cloud Networking (CloudNet)*, pages 339–343, 2014.

[58] Mostafa Rezvani, Mohammad Kazem Akbari, and Bahman Javadi. Resource allocation in cloud computing environments based on integer linear programming. *The Computer Journal*, 58(2):300–314, 2015.

[59] Qiaomin Xie, Xiaobo Dong, Yi Lu, and Rayadurgam Srikant. Power of d choices for large-scale bin packing: A loss model. *ACM SIGMETRICS Performance Evaluation Review*, 43(1):321–334, 2015.

[60] Arpan Mukhopadhyay, A Karthik, Ravi R Mazumdar, and Fabrice Guillemin. Mean field and propagation of chaos in multi-class heterogeneous loss models. *Performance Evaluation*, 91:117–131, 2015.

[61] Arpan Mukhopadhyay, Ravi R Mazumdar, and Fabrice Guillemin. The power of randomized routing in heterogeneous loss systems. In *Proceedings of the 27th International Teletraffic Congress (ITC)*, pages 125–133, 2015.

[62] Alexander Stolyar and Yuan Zhong. A service system with packing constraints: Greedy randomized algorithm achieving sublinear in scale optimality gap. *arXiv preprint arXiv:1511.03241*, 2015.

[63] Alexander L Stolyar and Yuan Zhong. An infinite server system with general packing constraints: Asymptotic optimality of a greedy randomized algorithm. In *Proceedings of the 51st Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 575–582, 2013.

[64] Saeed Javanmardi, Mohammad Shojafar, Danilo Amendola, Nicola Cordeschi, Hongbo Liu, and Ajith Abraham. Hybrid genetic algorithm for cloud computing applications. *arXiv preprint arXiv:1404.5528*, 2014.

[65] Cong Xu, Jiahai Yang, Di Fu, and Hui Zhang. Towards optimal collaboration of policies in the two-phase scheduling of cloud tasks. In *Proceedings of the IFIP International Conference on Network and Parallel Computing*, pages 306–320, 2014.

[66] Siva Theja Maguluri and R. Srikant. Scheduling jobs with unknown duration in clouds. *IEEE/ACM Transactions on Networking (TON)*, 22(6):1938–1951, 2014.

[67] Siva Theja Maguluri, R Srikant, and Lei Ying. Heavy traffic optimal resource allocation algorithms for cloud computing clusters. *Performance Evaluation*, 81:20–39, 2014.

[68] Leandros Tassiulas and Anthony Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE transactions on automatic control*, 37(12):1936–1948, 1992.

[69] Mihalis G Markakis, Eytan Modiano, and John N Tsitsiklis. Delay analysis of the max-weight policy under heavy-tailed traffic via fluid approximations. In *Proceedings of the 51st Annual Allerton Conference on Communication Control and Computing (Allerton)*, pages 436–444, 2013.

[70] Weifeng Sun, Ning Zhang, Haotian Wang, Wenjuan Yin, and Tie Qiu. Paco: A period aco based scheduling algorithm in cloud computing. In *Proceedings of the International Conference on Cloud Computing and Big Data (CloudCom-Asia)*, pages 482–486, 2013.

[71] Mihalis G Markakis, Eytan Modiano, and John N Tsitsiklis. Max-weight scheduling in queueing networks with heavy-tailed traffic. *IEEE/ACM Transactions on Networking (TON)*, 22(1):257–270, 2014.

[72] Javad Ghaderi. Randomized algorithms for scheduling vms in the cloud. In *Proceedings of the of International Conference on Computer Communications INFOCOM*, 2016.

[73] Pham Phuoc Hung, Mui Van Nguyen, Mohammad Aazam, and Eui-Nam Huh. Task scheduling for optimizing recovery time in cloud computing. In *Proceedings of the International Conference on Computing, Management and Telecommunications (ComManTel)*, pages 188–193, 2014.

[74] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.

[75] Sarang Dharmapurikar, Praveen Krishnamurthy, Todd Sproull, and John Lockwood. Deep packet inspection using parallel bloom filters. In *Proceedings of the 11th symposium on High performance interconnects, 2003.*, pages 44–51, 2003.

[76] Tania Lorido-Botrán, José Miguel-Alonso, and Jose Antonio Lozano. Auto-scaling techniques for elastic applications in cloud environments. *Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09*, 2012.

[77] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Proceedings of the International Conference on Cloud Computing (CLOUD),*, pages 500–507, 2011.

[78] Ming Mao and Marty Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12, 2011.

[79] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A Lozano. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, 12(4):559–592, 2014.

[80] Amazon elastic computing cloud. `http://aws.amazon.com/ec2`. Accessed 2016-07-29.

[81] Amazon cloudwatch developer guide api version 2010-08-01. `http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/acw-dg.pdf`,. Accessed 9th October, 2015.

[82] Amazon cloudwatch developer guide api version 2010-08-01. `http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/acw-dg.pdf`. Accessed 2015-10-9.

[83] A. D. George. How to Auto-scale an Application. `https://azure.microsoft.com/en-gb/documentation/articles/cloud-services-how-to-scale/`, 2011. Accessed 2015-10-9.

[84] Ibm cloud orchestrator version 2.4.0.2 user's guide. `http://www.ibm.com/support/knowledgecenter/SS4KMC_2.4.0.2/com.ibm.sco.doc_2.4/ICO_User_Guide.pdf`, 2015. Accessed 2016-07-29.

[85] Maria Abrahms. Rackspace auto scale overview. `https://support.rackspace.com/how-to/rackspace-auto-scale-overview/`. Accessed 2016-07-29.

[86] Susan Million. Install and configure the rackspace monitoring agent. `https://support.rackspace.com/how-to/install-and-configure-the-rackspace-monitoring-agent/`, 2015. Accessed 2016-07-29.

[87] Susan Million. Install and configure the cloud monitoring agent. `http://www.rackspace.com/knowledge_center/article/install-and-configure-the-cloud-monitoring-agent`, 2015. Accessed 9th October, 2015.

[88] Ming Mao and Marty Humphrey. A performance study on the vm startup time in the cloud. In *Proceedings of the 5th International Conference on Cloud Computing (CLOUD)*, pages 423–430, 2012.

[89] Elastic load balancing developer guide. `http://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/elb-ug.pdf`. Accessed 2015-10-9.

[90] Søren Asmussen. *Applied probability and queues*, volume 51. Springer Science & Business Media, 2008.

Appendix A

Data Sets

This appendix include values for the graphs plotted in this thesis.

Table A.1 Average Latency Centralized

Time Slot	Lamda	ScanOPT	ScanAll	ScanNone
200000	508.9537495	742.2210745	818.0639644	29912.00937
400000	513.5139763	780.6653145	905.8334974	59800.39053
600000	512.8637821	830.3156821	1010.084936	89800.57337
800000	517.6165387	911.6836216	1137.699229	119813.0487
1000000	517.7724021	1006.330005	1291.240047	149841.7933
1200000	520.0024258	1065.114661	1418.55936	179817.0337
1400000	519.1306663	1105.498118	1525.271178	209779.5107
1600000	521.5263258	1115.111395	1605.66698	239779.0768
1800000	521.575393	1128.203	1687.934356	269813.7497
2000000	520.0963925	1125.197607	1744.134539	299762.4867
2200000	519.8998475	1120.56208	1797.734032	329758.2874
2400000	520.646736	1122.727934	1857.787334	359748.143
2600000	519.99109	1128.888324	1922.827769	389718.155
2800000	519.3205393	1126.041902	1976.219581	419708.3173
3000000	519.4143654	1115.467046	2023.586108	449688.7146
3200000	519.7155181	1109.757721	2075.998202	479680.2757
3400000	519.0464457	1115.480342	2138.438229	509656.1005
3600000	518.0514835	1109.400254	2179.017652	539657.9764
3800000	518.1101033	1106.061253	2219.748768	569650.7176
4000000	518.0651797	1102.781768	2254.836303	599625.3496

Table A.2 Maximum Latency Centralized

Time Slot	Lamda	ScanOPT	ScanAll	ScanNone
200000	2979.2	3762.3	3989.9	140607.6
400000	3134.1	4118.8	4542.9	279711.3
600000	3170.1	4411.6	5039.5	418735.3
800000	3246.9	5036	5938.6	557225.3
1000000	3319.3	5470.6	6591.1	695508.8
1200000	3384.5	5834.2	7150.1	833855.2
1400000	3384.5	5931.7	7535.4	972667.5
1600000	3485.6	6141	7915.1	1112606.5
1800000	3529.4	6420.3	8377.3	1251215.2
2000000	3529.4	6420.3	8526.7	1390035.3
2200000	3539.9	6553.2	8843.1	1528112
2400000	3539.9	6589.4	9084.9	1666022.3
2600000	3539.9	6616.2	9251	1805216.9
2800000	3539.9	6629.8	9395.7	1942639.8
3000000	3543.3	6629.8	9618.9	2081929.4
3200000	3544.6	6629.8	9826.4	2221191.6
3400000	3544.6	6629.8	10189.3	2361424.2
3600000	3544.6	6629.8	10242.5	2500488.6
3800000	3544.6	6678.2	10521	2638843.9
4000000	3544.6	6678.2	10662.2	2777764.3

Table A.3 Latency Differences and Ratios Centralized

Time Slot	AveRatio	MaxRatio	AveDiff	MaxDiff
200000	1.102183692	1.06049491	75.84288993	227.6
400000	1.160335269	1.102966884	125.1681829	424.1
600000	1.216507116	1.142329314	179.7692541	627.9
800000	1.247910133	1.179229547	226.0156076	902.6
1000000	1.283117904	1.20482214	284.9100418	1120.5
1200000	1.33183723	1.225549347	353.4446989	1315.9
1400000	1.379713953	1.270360942	419.7730602	1603.7
1600000	1.439916215	1.288894317	490.5555848	1774.1
1800000	1.496126457	1.304814417	559.7313567	1957
2000000	1.550069542	1.328084357	618.9369324	2106.4
2200000	1.604314534	1.349432338	677.1719512	2289.9
2400000	1.654708392	1.378714299	735.0594004	2495.5
2600000	1.703293168	1.398234636	793.9394451	2634.8
2800000	1.755014247	1.417192072	850.1776792	2765.9
3000000	1.814115544	1.450858246	908.1190613	2989.1
3200000	1.870676962	1.482156324	966.2404815	3196.6
3400000	1.917055951	1.536894024	1022.957886	3559.5
3600000	1.964140214	1.544918399	1069.617398	3612.7
3800000	2.006894972	1.575424516	1113.687515	3842.8
4000000	2.044680433	1.596567937	1152.054535	3984

Table A.4 Average Queue Sizes Centralized

Time Slot	Lamda	ScanOPT	ScanAll	ScanNone
200000	256.6152895	416.597637	464.285194	17994.11417
400000	258.9904947	440.569204	519.329507	35829.39166
600000	258.5349542	471.4936707	584.5707503	53744.22788
800000	261.0139995	522.794109	665.0624911	71681.05329
1000000	261.0697983	582.2961065	761.7189776	89610.12394
1200000	262.1905491	619.1937198	841.7752223	107526.8736
1400000	261.7253521	644.4557983	908.7396621	125427.3642
1600000	262.9483935	650.5028473	959.3717079	143339.2111
1800000	262.9470899	658.6682481	1011.055447	161264.2811
2000000	262.2086057	656.7094	1046.29864	179178.8182
2200000	262.116159	653.7799066	1080.009596	197085.4752
2400000	262.5200964	655.2048229	1117.894149	214987.6004
2600000	262.1686577	659.0767742	1158.850053	232890.4523
2800000	261.8314039	657.2879982	1192.448996	250791.4502
3000000	261.8700955	650.6110189	1222.203664	268690.2356
3200000	262.0376092	647.0398852	1255.187095	286596.6074
3400000	261.6870985	650.6357546	1294.473521	304516.4022
3600000	261.1733477	646.7894689	1319.982101	322435.0823
3800000	261.2037882	644.6701289	1345.558964	340353.3991
4000000	261.1834657	642.6208019	1367.682144	358260.2409

Table A.5 Maximum Queue Centralized

Time Slot	Lamda	ScanOPT	ScanAll	ScanNone
200000	459.1	889.8	988.7	35878.4
400000	503.4	1063.1	1242.3	71769.6
600000	512.3	1180.2	1445.7	107684.6
800000	532.4	1392.5	1764.9	143665.5
1000000	550	1611.4	2053.8	179345.6
1200000	570.3	1781	2338.2	215179.6
1400000	570.3	1813.3	2463.6	251001.5
1600000	593.4	1908.3	2615.6	286981.3
1800000	610.4	2017.8	2830	322646.2
2000000	610.4	2017.8	2869.6	358496.7
2200000	610.7	2037.1	2938.9	394330.3
2400000	610.7	2068.9	3041.8	430302.1
2600000	610.7	2068.9	3123.5	466168.9
2800000	610.7	2109.2	3207.9	501926.9
3000000	610.7	2109.2	3288.6	537761.8
3200000	613	2109.2	3360.9	573887.8
3400000	613	2109.2	3474.2	609840.7
3600000	613	2109.2	3526.3	645436.8
3800000	613	2125.6	3628	681054.3
4000000	615.6	2125.6	3721	716771.1

Table A.6 Queue Differences and Ratios

Time Slot	AveRatio	MaxRatio	AveDiff	MaxDiff
200000	1.114469101	1.111148573	47.687557	98.9
400000	1.178769424	1.168563635	78.760303	179.2
600000	1.239827354	1.224961871	113.0770797	265.5
800000	1.272130806	1.267432675	142.2683821	372.4
1000000	1.308129952	1.274543875	179.4228711	442.4
1200000	1.359469897	1.312857945	222.5815026	557.2
1400000	1.410088426	1.358627916	264.2838639	650.3
1600000	1.47481554	1.370644029	308.8688607	707.3
1800000	1.534999523	1.402517593	352.3871987	812.2
2000000	1.593244501	1.422142928	389.5892403	851.8
2200000	1.651946756	1.442688135	426.2296896	901.8
2400000	1.706175092	1.470249891	462.6893262	972.9
2600000	1.758292962	1.509739475	499.7732791	1054.6
2800000	1.814195603	1.520908401	535.1609978	1098.7
3000000	1.878547441	1.559169353	571.5926454	1179.4
3200000	1.939891379	1.593447753	608.1472102	1251.7
3400000	1.989551776	1.647164802	643.8377667	1365
3600000	2.040821882	1.67186611	673.1926321	1417.1
3800000	2.087205383	1.706812194	700.8888348	1502.4
4000000	2.128288003	1.750564546	725.0613416	1595.4

Table A.7 Maximum Latency Decentralized

Time Slot	OPT_P2Q	OPT_P2W	OPT_UR	OPT_JSQ	OPT_JSW	OPT_RR
200000	24915.7	7932.8	39021.3	49130.8	7668.7	35776.9
400000	31178.4	8773.4	52604	83312.1	8517.3	50933.9
600000	38318.7	9401.1	70550.1	121819	9322.7	60500.4
800000	44405.6	10426	83393.8	154086.6	10199.3	75868
1000000	50769.7	10508.1	91588.1	196281.4	10610.9	84110.1
1200000	54445.5	10862.6	99762.8	206655.4	11020.3	96393.2
1400000	66501.1	11311.8	107913.1	233907.4	11675.7	104475.3
1600000	70114.6	12000	110042.7	270911.3	12118.4	113457.8
1800000	75558.9	12237.3	114007.1	296374.6	12126.1	128655.1
2000000	83571.8	12787.4	120899	313306.1	12363	135208.9
2200000	85431.9	12787.4	126970.2	340073.2	12669.7	141216.1
2400000	86995.9	12828.3	129434.7	363831	12669.7	150663.2
2600000	87687.2	12828.3	133355.4	383831	12692.1	154965.2
2800000	98935.8	13181.3	140550	408569.7	12875.8	161744.8
3000000	100526.6	13302.8	147784.8	450107.1	13236.3	169616.3
3200000	108846.3	13793.3	153317.3	488108.8	13407.2	177577.4
3400000	109053.2	13844.1	153959.5	506422.7	13407.2	187101.6
3600000	110701	13984.4	158809.3	543386.1	13456.6	198711.9
3800000	113501.9	14143	160376.9	589772.1	14132.3	201616.3
4000000	114465.6	14317.8	164852.8	610776.2	14132.3	208664.9

Table A.8 Average Latency Decentralized

Time Slot	OPT_P2Q	OPT_P2W	OPT_UR	OPT_JSQ	OPT_JSW	OPT_RR
200000	1325.056606	932.6141992	3225.941305	749.4007685	850.873193	3279.403379
400000	1465.980664	1006.823605	4414.867406	829.4268213	908.5613424	4774.292229
600000	1585.927951	1079.632336	5360.544497	912.8895094	967.9743923	6117.989342
800000	1685.2803	1148.348811	6166.620935	994.0642323	1027.210989	7365.895597
1000000	1783.063473	1220.771833	6880.373287	1086.893236	1092.280084	8566.980737
1200000	1831.354033	1255.658238	7504.019741	1129.507554	1123.132161	9695.093556
1400000	1874.569797	1288.463303	8075.697481	1169.452507	1152.926967	10777.80058
1600000	1934.458796	1334.721743	8623.970164	1228.66596	1196.639539	11868.93865
1800000	1992.852787	1385.321099	9152.713192	1296.910676	1247.203059	12944.16732
2000000	2045.024192	1430.160338	9640.963656	1357.830404	1291.643766	13991.02025
2200000	2072.228401	1451.520789	10086.43031	1385.714326	1311.990413	15000.75371
2400000	2090.616961	1463.678932	10510.42189	1400.363338	1322.892208	15990.78669
2600000	2093.900509	1462.749572	10897.38944	1396.28219	1320.25374	16960.06477
2800000	2115.835988	1477.142403	11284.35455	1412.520962	1332.052517	17939.90871
3000000	2147.606164	1500.325389	11670.62448	1440.38878	1352.882715	18929.89974
3200000	2188.849424	1532.486123	12056.68977	1480.842897	1383.060268	19934.0999
3400000	2215.180408	1552.039355	12412.15932	1505.565364	1401.165866	20919.11531
3600000	2240.210002	1570.958414	12758.44804	1530.373496	1418.88669	21901.4068
3800000	2253.926447	1580.512544	13093.98052	1541.904188	1427.600817	22877.01016
4000000	2260.185067	1584.365798	13413.60162	1545.952041	1430.410007	23840.57066

Table A.9 Maximum Queue Sizes Decentralized

Time Slot	OPT_P2Q	OPT_P2W	OPT_UR	OPT_JSQ	OPT_JSW	OPT_RR
200000	1386.7	1019.5	3843.2	889.1	906.2	4099.1
400000	1693.2	1200.4	5451.9	1108.1	1086.9	6361.9
600000	1886.5	1356	6705.7	1259.1	1213.3	8304.5
800000	2130.4	1590.8	7825.9	1521.4	1438.6	10285.5
1000000	2281.3	1711.7	8661.4	1651.1	1570.1	12076.5
1200000	2371.2	1801.8	9470.3	1758.4	1653	13733.7
1400000	2573.3	2005	10167.2	2009.8	1860.6	15481.5
1600000	2717.1	2138.7	10892.8	2172.5	1993.1	17140.3
1800000	2796.6	2206.3	11671.9	2253.9	2062.9	18855.7
2000000	2868.2	2276.5	12189.7	2328.4	2129.4	20370.6
2200000	2868.2	2276.8	12632.5	2337.2	2129.4	21724
2400000	2883.2	2291.9	13138.9	2359.3	2140.8	23343.9
2600000	2924.3	2322.9	13456.4	2397.6	2174.7	24783.6
2800000	2986	2382.6	14137.2	2458.2	2232.5	26584.9
3000000	3064.9	2450.4	14763.3	2535.3	2293.9	28305.7
3200000	3148.2	2534.5	15314.2	2638.2	2374.4	29979.9
3400000	3154.7	2534.5	15706.3	2638.3	2374.4	31555
3600000	3196.2	2566.6	16154	2668.8	2399.3	33260
3800000	3312.2	2684.2	16584	2806.9	2513.5	34678.7
4000000	3327.6	2700.7	16811.4	2807	2521.9	36085.5

Table A.10 Average Queue Sizes Decentralized

Time Slot	OPT_P2Q	OPT_P2W	OPT_UR	OPT_JSQ	OPT_JSW	OPT_RR
200000	792.6151505	545.2538555	1990.208472	428.3609745	489.095815	2022.72288
400000	880.6564125	591.7166265	2736.570506	477.7554742	525.1227098	2961.31818
600000	955.2964405	637.1896715	3331.024256	529.0694852	562.188367	3805.709321
800000	1017.445299	680.3567321	3838.29464	579.2298423	599.4177792	4590.365344
1000000	1078.537838	725.7977939	4286.881335	636.2936019	640.2449423	5345.285822
1200000	1108.786305	747.7704612	4679.125469	662.6189658	659.6781547	6054.577992
1400000	1135.897274	768.4370672	5038.898166	687.3748366	678.4637834	6736.542065
1600000	1173.263398	797.4553886	5383.525612	724.0186296	705.8977864	7422.408951
1800000	1209.865079	829.3427283	5716.105475	766.253567	737.7564017	8098.580107
2000000	1242.349375	857.4719896	6022.704516	803.9223647	765.6230491	8756.52041
2200000	1259.240355	870.8136982	6302.200763	821.1477793	778.335931	9390.685274
2400000	1270.832788	878.5230987	6568.890598	830.2737817	785.2351668	10014.07825
2600000	1272.847485	877.9124665	6812.297955	827.7574295	783.5567387	10624.03515
2800000	1286.655024	887.0314058	7056.001778	837.8654858	791.0329616	11240.71304
3000000	1306.639393	901.7032363	7299.654036	855.1903761	804.2104199	11864.45735
3200000	1332.379081	921.8969914	7542.39643	880.2844836	823.1713167	12495.97604
3400000	1348.788912	934.1379599	7765.63545	895.5330652	834.5118471	13115.08864
3600000	1364.399733	946.0265097	7983.516901	910.8118987	845.6417342	13732.81028
3800000	1372.921147	951.9959225	8194.333357	917.8695453	851.0960225	14346.31873
4000000	1376.804661	954.3865978	8395.355654	920.3174193	852.8260075	14952.50536

Table A.11 Maximum Latency Size Non-preemptive

Time Slot	OPT_P2Q	OPT_P2W	OPT_UR	OPT_JSQ	OPT_JSW	OPT_RR
200000	64942.8	57682.7	50282.2	139994.1	115345	48390.2
400000	80812.7	71642.3	71279	189901	120963.2	67834.1
600000	88176.8	84646.3	88165	232908.7	129053.9	85110.9
800000	105836.7	90604	100730	310165.4	143267.5	105713.3
1000000	112563.9	96623.7	106737.5	374439.1	158040.5	117345
1200000	115192.4	100736.2	115991.6	403919.2	162827.7	124969.9
1400000	117063.9	102885.9	126646.1	442179	166279.2	129974.7
1600000	121012	103615.8	132825.6	489455.7	182195.5	144857.9
1800000	122579.6	104093.8	147567.4	532528.9	182195.5	151356.1
2000000	127001.3	104637.8	152986.9	576366	183407.1	154534.4
2200000	128753.9	106972.1	159076.7	641341.8	184886.2	161556.3
2400000	130011.3	109217.9	168618.3	718573.3	190935.7	182100.3
2600000	130898.5	109593.8	168838.8	741227.2	190935.7	187473.1
2800000	134027.7	112682.4	173039.3	779297.4	196463.1	194576.5
3000000	141244.2	114003.1	174709	847889	201301.8	198908.2
3200000	147328.9	114786.6	176302.8	895231.1	203236.9	206400
3400000	149794	115332.5	183335.6	947876.3	204283.5	216397.5
3600000	154178.6	115332.5	192880	973528	204283.5	220547.9
3800000	155505.3	116452.3	194341.8	1003478	207040.9	226381.1
4000000	158485.9	117662.9	197972.6	1018727	207845.3	231696.7

Table A.12 Average Latency Size Non-preemptive

Time Slot	OPT_P2Q	OPT_P2W	OPT_UR	OPT_JSQ	OPT_JSW	OPT_RR
200000	2680.657746	2669.133772	4318.157379	823.8276227	1157.467446	4468.164291
400000	3429.757198	3434.257414	5935.665342	939.3708458	1305.55756	6480.225775
600000	4017.873961	4044.910577	7194.901488	1055.997143	1450.808992	8199.362621
800000	4508.72903	4539.844042	8256.570346	1159.956899	1582.382841	9745.418145
1000000	4933.469027	4974.368335	9190.746411	1263.073296	1697.31952	11196.48949
1200000	5276.836765	5325.087341	9985.556182	1315.056073	1757.409998	12539.69395
1400000	5571.544985	5623.545904	10690.25121	1364.182868	1802.5878	13807.83576
1600000	5863.389836	5918.742658	11364.86903	1431.93935	1859.740911	15059.00359
1800000	6134.891048	6193.440732	11992.59423	1495.67751	1920.054712	16279.66206
2000000	6373.172039	6434.283552	12573.34482	1552.556625	1973.627344	17455.85606
2200000	6571.95434	6637.765401	13100.5957	1581.917604	2008.580058	18584.85882
2400000	6751.117242	6819.057944	13592.9126	1604.402101	2034.103762	19685.08162
2600000	6904.194848	6975.617687	14050.29544	1611.16009	2045.177005	20754.4536
2800000	7070.051142	7143.625548	14509.02887	1637.29347	2073.802569	21824.87538
3000000	7245.113941	7319.380413	14968.37445	1673.491767	2112.111906	22901.51512
3200000	7428.988344	7502.931119	15430.858	1720.024657	2158.415526	23984.84085
3400000	7592.562313	7666.338686	15864.76969	1750.807917	2189.707452	25045.96115
3600000	7751.786119	7824.986108	16285.42608	1779.435779	2220.652754	26098.67508
3800000	7897.54912	7971.543185	16685.68579	1796.369097	2243.019215	27140.1434
4000000	8029.875073	8105.372698	17065.88868	1805.530113	2255.516818	28167.18419

Table A.13 Maximum Queue Size Non-preemptive

Time Slot	OPT_P2Q	OPT_P2W	OPT_UR	OPT_JSQ	OPT_JSW	OPT_RR
200000	3515.2	3378.8	6184.1	1413.4	1727	6560.1
400000	4562.4	4619.1	8594.3	1673.2	2161.3	9994
600000	5661.1	5664.7	10643	1990.7	2543.8	13276.2
800000	6453.3	6422.7	12317.2	2315.8	2894.6	15706.1
1000000	6904	7007	13937.3	2462.3	2937.6	18263.4
1200000	7351.3	7574.7	14883	2626.7	3135.7	20376.3
1400000	7861.6	8138	15928.7	2861.4	3424.7	22676.8
1600000	8332.1	8520.8	16952.8	3040.2	3517.9	25145.5
1800000	8764.9	8960.7	17780.3	3121.7	3733	27663.1
2000000	9129.5	9269.5	18753.1	3244.5	3827.1	29239.7
2200000	9256.6	9340.3	19204.4	3244.5	3827.1	31610.6
2400000	9513.7	9499.8	19897.7	3274.4	3827.1	33558.2
2600000	9616.8	9518.1	20432	3335	3827.1	35089.2
2800000	10076.2	9998.6	21438.3	3383.1	3911.1	36916.2
3000000	10537.5	10692.9	22317.5	3440.5	3997.2	38650.1
3200000	10779.6	10899.3	23142.9	3546.7	4089.8	41215.2
3400000	11171.2	11255.8	23824.8	3558.5	4121.8	43081.5
3600000	11458.2	11526.9	24589.8	3639.5	4226.3	45119.5
3800000	11592	11788.7	25337.7	3789.4	4346.1	46683.1
4000000	11802.4	11865.9	25969.5	3807.8	4412.8	48669.6

Table A.14 Average Queue Size Non-preemptive

Time Slot	OPT_P2Q	OPT_P2W	OPT_UR	OPT_JSQ	OPT_JSW	OPT_RR
200000	1659.000626	1647.589319	2732.645288	506.1055465	716.8869465	2832.930043
400000	2127.700385	2125.492148	3771.402859	579.5079648	812.0733085	4128.939016
600000	2497.808959	2509.075551	4583.458726	654.3250778	906.591403	5239.673554
800000	2807.071896	2820.447311	5270.077983	720.4864011	992.1917321	6238.868532
1000000	3076.979828	3095.83567	5874.817442	786.2900617	1066.408157	7174.095482
1200000	3294.408762	3318.364068	6389.31222	819.5411862	1106.079689	8042.831782
1400000	3481.561812	3509.160609	6846.076487	850.8987068	1136.094907	8863.693882
1600000	3667.134211	3697.118492	7281.258564	893.9717427	1172.936164	9676.233279
1800000	3840.666383	3872.751815	7686.669288	934.3633339	1212.006625	10466.03871
2000000	3993.215868	4027.134896	8063.533941	970.3867282	1246.254268	11225.88719
2200000	4119.267716	4157.635228	8404.217404	988.9666833	1268.542233	11956.31631
2400000	4234.64118	4274.325062	8723.20354	1003.344109	1285.021224	12671.76244
2600000	4332.791446	4374.62089	9019.321218	1007.661781	1292.07349	13365.05921
2800000	4439.550119	4482.914229	9316.980533	1024.493104	1310.538005	14057.43516
3000000	4552.553541	4596.639719	9614.332049	1047.778417	1335.366641	14752.73976
3200000	4670.30911	4715.091572	9913.240023	1077.439627	1365.352348	15450.80796
3400000	4775.528667	4819.690756	10193.71151	1096.901836	1385.497034	16135.72292
3600000	4878.082272	4922.351218	10465.53815	1115.019135	1405.414851	16814.39685
3800000	4971.630491	5017.355321	10723.56075	1125.859038	1419.807175	17486.28267
4000000	5057.187339	5104.309936	10970.71077	1131.591334	1427.893806	18148.73327

Table A.15 Maximum Latency Heterogeneous 1

Time Slot	OPT_P2Q	OPT_P2W	OPT_UR	OPT_JSQ	OPT_JSW	OPT_RR
200000	44722.8	10857.7	72381	66616.5	12259.4	68587.7
400000	52784.4	11894.5	127770.6	89501.1	12883.9	126701.7
600000	60431.4	12534.7	171762.4	95127.5	13278.9	175381.2
800000	66405	12782	222749.3	102211.8	13526.4	227385.7
1000000	67119.3	12782	273842.3	102927.3	13526.4	275748.2
1200000	67119.3	12956.8	323947.5	106285.6	13556.4	327847.1
1400000	68080.2	13089.5	367542.9	107469.3	13556.4	379850.7
1600000	82474.9	13260.1	411158.1	108088.5	13556.4	427613.7
1800000	82474.9	13318.1	452417.7	108088.5	13670.7	478305.8
2000000	83757.7	13403.2	494466.6	111630.4	13798.4	520566.1
2200000	84754.3	13575.1	536804.2	111809.8	13798.4	569578.5
2400000	84985.9	13613.5	582467	111809.8	13798.4	619165.1
2600000	85127.2	13613.5	628889.5	111809.8	13802.2	674542.7
2800000	87419.9	13746.9	678765.2	112412.2	13893	718321.1
3000000	88933	13746.9	729299.1	114428.9	14093.2	772252.1
3200000	90790	13866.3	783589.2	115051.7	14093.2	828383.8
3400000	90790	13866.3	831481.8	115687.8	14093.2	874495.3
3600000	90790	13896.9	868191.6	116773.3	14167.9	915024.1
3800000	94355.8	13915	907464.4	116773.3	14167.9	960544.6
4000000	94355.8	13915	950637.5	122169.4	14227.2	1010246

Table A.16 Average Latency Heterogeneous 1

Time Slot	OPT_P2Q	nOPT_P2W	OPT_UR	OPT_JSQ	OPT_JSW	OPT_RR
200000	1669.918985	1464.603315	9620.111056	778.9133299	577.5752038	10341.78447
400000	1811.369842	1596.662383	18323.54043	800.7341248	580.7410224	20021.90245
600000	1901.016739	1669.044788	27105.8226	805.9032133	582.38745	29782.67544
800000	1959.80164	1711.220789	35883.90368	814.9371473	583.3852832	39578.89393
1000000	1997.76216	1740.68724	44716.86467	817.3520459	583.8970939	49432.61812
1200000	2010.580898	1750.120631	53508.44426	820.5728656	583.9846124	59243.7457
1400000	2022.964741	1756.840779	62312.41397	825.1068878	584.455218	69057.03383
1600000	2038.279131	1770.351793	71146.50707	824.4813011	584.6056531	78889.79132
1800000	2052.994075	1784.790432	79988.41093	826.8347504	585.0309495	88730.43039
2000000	2056.849643	1788.557552	88842.8234	825.5812296	584.960737	98586.68575
2200000	2049.221501	1785.662434	97665.34402	824.4656445	584.8347998	108415.5084
2400000	2050.928223	1787.089231	106474.5361	824.4335394	584.8178782	118231.8128
2600000	2050.360736	1784.004343	115275.4273	823.5112236	584.9075654	128031.4339
2800000	2064.37461	1791.878334	124077.5106	826.1301811	585.3294365	137819.1787
3000000	2076.446309	1800.138878	132902.3859	828.4209027	585.7715165	147632.9417
3200000	2086.791655	1805.564147	141755.6455	829.4033232	585.8616847	157489.3159
3400000	2087.641202	1804.143267	150604.9313	829.2092296	585.9289236	167343.9995
3600000	2092.689422	1807.34168	159435.3194	830.1744012	586.0624836	177185.716
3800000	2092.17724	1807.468576	168269.7977	829.7416241	586.024568	187032.4426
4000000	2091.163764	1807.364756	177095.6833	829.3850731	586.0160319	196873.4768

Table A.17 Maximum Queue Heterogeneous 1

Time Slot	OPT_P2Q	OPT_P2W	OPT_UR	OPT_JSQ	OPT_JSW	OPT_RR
200000	1711	1526.9	15183	879.9	531.9	16599.5
400000	1929.1	1751.2	29767.7	961.6	545.4	32888.9
600000	2062.2	1852.4	44582.6	986.2	552	49298.9
800000	2172.6	1943.8	59524.9	1044.7	562	65956
1000000	2212.9	1969.5	74315.3	1045.1	564.7	82512.6
1200000	2287.3	2018	88910.2	1071	567.2	98671.2
1400000	2342.3	2069.7	103615	1120.5	572.3	115061.2
1600000	2355.4	2102.5	118516.8	1134.8	572.3	131650.5
1800000	2381.6	2123.8	133208.8	1134.8	580.6	147979.2
2000000	2381.6	2137.6	147791.2	1145.3	582	164354
2200000	2381.6	2137.6	162325.1	1151.8	585.7	180493.3
2400000	2381.6	2138.6	177297.4	1168.9	590.4	197192.4
2600000	2400.3	2159.2	192032.7	1168.9	596.7	213471.3
2800000	2414.6	2159.6	206858.6	1194.3	597.2	229708.2
3000000	2421.3	2171.1	221761.1	1194.3	600.2	246581.4
3200000	2468.7	2204.4	236627.3	1238.4	613	263037.6
3400000	2468.7	2204.4	251014.3	1238.4	617.6	279343.1
3600000	2486.3	2213	266121.5	1239.1	617.6	296129.5
3800000	2492.4	2224	280953.8	1239.1	617.6	312565.7
4000000	2492.4	2224	295677.6	1239.1	623.7	329150.7

Table A.18 Average Queue Heterogeneous 1

Time Slot	OPT_P2Q	OPT_P2W	OPT_UR	OPT_JSQ	OPT_JSW	OPT_RR
200000	1007.942909	880.2900735	6012.298103	450.4090635	323.2451375	6466.474535
400000	1096.661398	963.1808565	11484.42807	464.1484605	325.2349105	12550.33224
600000	1152.764743	1008.597193	17005.15782	467.27298	326.1657873	18688.94442
800000	1189.602875	1035.143656	22528.16024	472.8787893	326.796818	24849.97957
1000000	1213.234989	1053.63248	28080.26484	474.3160962	327.0973999	31043.86895
1200000	1221.283582	1059.660644	33611.73804	476.3656166	327.1938018	37215.27327
1400000	1229.253393	1063.981062	39151.80581	479.2521432	327.5108096	43390.59655
1600000	1238.799086	1072.424378	44706.85799	478.8577053	327.5668029	49573.3837
1800000	1248.060696	1081.590225	50267.06869	480.3486959	327.8582133	55761.36074
2000000	1250.391503	1083.868664	55828.91535	479.5188212	327.7951746	61953.01093
2200000	1245.48667	1081.92023	61370.29513	478.7281746	327.6894664	68127.61885
2400000	1246.633949	1082.897448	66911.84288	478.7307323	327.7038871	74303.06755
2600000	1246.275425	1080.948753	72449.48217	478.1863991	327.7566195	80471.62111
2800000	1255.167576	1086.033511	77990.35872	479.9049503	328.0669403	86634.95317
3000000	1262.848735	1091.354623	83550.56985	481.4096234	328.3781782	92816.98665
3200000	1269.333042	1094.768171	89112.8492	482.0078245	328.4272463	99007.89332
3400000	1269.793587	1093.831956	94676.88017	481.866558	328.4525648	105204.0022
3600000	1272.987273	1095.866651	100230.2739	482.4724707	328.5525742	111393.151
3800000	1272.630329	1095.926023	105785.429	482.1923595	328.5087466	117582.662
4000000	1271.972068	1095.839622	111338.5831	481.9885789	328.4954137	123773.3047

Table A.19 Maximum Latency Heterogeneous c=0.97

Time Slot	P2Q	P2W	UR	JSQ	JSW	RR
200000	44132.4	9184	56947.6	96911.5	10299.1	56339.1
400000	50411.5	10045.2	99075.4	109265.2	10566.4	99269.5
600000	62696.3	11374.7	135639.1	120997.6	11013.8	139297.2
800000	66243.5	11689.7	178036.1	126712.5	11027.7	176450.8
1000000	73455.8	11865.6	214382.5	130533.7	11231.3	217688
1200000	85975.7	11948.6	253861	133957.6	11401.5	256578.2
1400000	88256.7	12122.7	288555.3	143785.2	11401.5	294671.5
1600000	94054	12287.4	328990.8	160217.1	11403.9	332046
1800000	95897.2	12426.9	356338.8	163582.2	11403.9	371023.7
2000000	99979.4	12533.3	391478.8	164507.4	11470.5	414918.9
2200000	102544.4	12627.9	431934.9	164507.4	11626.9	451248.6
2400000	107529.2	12763.6	464704.7	164507.4	11626.9	485864.4
2600000	110560.4	13033.7	501017.5	172906.9	11626.9	521378.7
2800000	117049	13335.7	536064.4	172906.9	11665.4	562557.5
3000000	121580.7	13437.3	576819.9	175286.1	11709.4	603345.4
3200000	121580.7	13530.5	605730.3	175334.2	11718.5	637009
3400000	121580.7	13551	643994.3	175654.9	11718.5	681399
3600000	122688	13567.2	681849	176134.1	11794.4	717523.3
3800000	122688	13744.4	719929.9	176134.1	11794.4	757414.7
4000000	122688	13960.3	757285.2	176134.1	12020	793803.2

Table A.20 Average Latency Heterogeneous c=0.97

Time Slot	P2Q	P2W	UR	JSQ	JSW	RR
200000	2092.293346	1779.5471	10340.39569	943.7438469	615.3156162	11091.19014
400000	2420.533528	2057.964634	19897.14351	946.1909896	618.0370368	21659.81487
600000	2623.071389	2238.066539	29440.04712	974.1606472	620.5187633	32211.35508
800000	2760.920402	2367.635598	39000.08484	979.8183984	620.7791562	42771.11462
1000000	2850.873668	2428.325001	48536.59239	977.970503	620.8206877	53293.69892
1200000	2930.851084	2486.328405	58085.00622	987.5350533	621.2108738	63823.11804
1400000	2974.743029	2500.407771	67615.8097	981.9993833	620.4555444	74336.57369
1600000	3004.029088	2511.976945	77111.31037	986.3300778	621.1265164	84816.18586
1800000	3043.864721	2545.92505	86632.03664	992.0062266	621.8606659	95324.09797
2000000	3086.311891	2578.501391	96180.46705	991.0911614	622.0428467	105860.0539
2200000	3115.721287	2603.83082	105747.6145	990.1131495	622.2322473	116417.0761
2400000	3144.847465	2625.902959	115294.3397	994.0384479	622.8938488	126952.1363
2600000	3190.510144	2667.083414	124851.5107	1001.135901	623.1663334	137501.6251
2800000	3231.038669	2698.467417	134437.1761	1000.735428	623.19138	148080.1709
3000000	3251.893358	2711.959157	144021.136	998.5795097	622.9316167	158659.6137
3200000	3260.329612	2720.619305	153604.1494	997.4855521	622.9028917	169241.217
3400000	3275.66846	2733.868435	163178.4874	1000.702521	623.1644572	179814.341
3600000	3290.481411	2745.408489	172750.4925	998.4282523	623.0529016	190386.6424
3800000	3300.76036	2750.161612	182336.8447	997.6734921	622.9580545	200969.8902
4000000	3317.9324	2761.507502	191918.5409	999.7419975	623.1799897	211544.7629

Table A.21 Maximum Queue Heterogeneous c=0.97

Time Slot	P2Q	P2W	UR	JSQ	JSW	RR
200000	1602.3	1340.9	10494.1	800.5	467.1	11360
400000	1793.3	1505.8	20219.9	836.5	475.3	22162.8
600000	2033.4	1777.1	30230.5	948.3	491.8	33159.5
800000	2111.8	1838.9	40196.2	972.3	495.6	44083.4
1000000	2128.6	1838.9	49863.5	1005.8	497.4	54910.5
1200000	2213.8	1881.6	59598.2	1023.1	499	65732.9
1400000	2250.8	1919	69326	1023.7	502.3	76345.2
1600000	2267.7	1927	79121.5	1032.6	504.2	87132.4
1800000	2326.9	1969	89250.4	1037.3	505.4	98305.2
2000000	2367.9	1969	99265.1	1037.3	507.4	109423.2
2200000	2391.5	2010.5	108927.5	1057.8	508	120111.5
2400000	2425.3	2051.6	118860.5	1057.8	509.8	130964.1
2600000	2487.2	2123.5	128802.6	1074.6	511.2	141942.6
2800000	2531	2162.7	138788.5	1083.9	511.2	153022.3
3000000	2559.3	2213.7	148666.5	1083.9	511.2	163894
3200000	2563.9	2222.4	158459.2	1083.9	511.2	174706.5
3400000	2619.9	2239	168413.2	1085.6	518	185810
3600000	2619.9	2239	178262.9	1085.6	520.7	196677.4
3800000	2619.9	2239	188384.2	1085.6	526.1	207679.5
4000000	2646.6	2273.6	198077.6	1095.1	527.8	218438.7

Table A.22 Average Queue Heterogeneous c=0.97

Time Slot	P2Q	P2W	UR	JSQ	JSW	RR
200000	1076.395744	915.344251	5322.559009	485.0494205	315.9323725	5709.190256
400000	1244.868552	1058.273611	10238.66906	486.0953112	317.2083253	11145.82696
600000	1349.80342	1151.587639	15157.64936	500.7781367	318.6559732	16584.62752
800000	1421.162411	1218.631835	20085.17636	503.807954	318.8689714	22027.32274
1000000	1467.610066	1250.006666	24998.46447	502.8874339	318.9123235	27448.68468
1200000	1508.742324	1279.831547	29914.67187	507.7878661	319.0944312	32870.02955
1400000	1530.835297	1286.628241	34811.98748	504.7631813	318.6009336	38272.3217
1600000	1545.815258	1292.493637	39698.76653	506.9627406	318.9294811	43665.58356
1800000	1566.574266	1310.172596	44607.49924	509.9638371	319.3582172	49083.257
2000000	1588.71316	1327.174674	49532.65564	509.5794411	319.5066876	54517.74099
2200000	1603.876935	1340.227755	54461.00983	509.0848539	319.6114515	59955.99968
2400000	1618.944078	1351.650866	59380.29903	511.1280492	319.9660016	65384.54966
2600000	1642.525321	1372.913072	64305.37408	514.8037418	320.1199888	70820.99334
2800000	1663.322401	1389.014373	69238.603	514.5718511	320.1145749	76265.21847
3000000	1674.025829	1395.934334	74172.70432	513.447485	319.9724067	81711.82895
3200000	1678.308324	1400.348037	79104.94495	512.8637625	319.9447869	87158.03331
3400000	1686.29954	1407.251551	84039.72157	514.5456427	320.0947232	92607.55761
3600000	1693.972376	1413.229459	88971.61766	513.3856912	320.0446702	98054.81484
3800000	1699.188381	1415.612423	93904.877	512.9731071	319.9818263	103501.0821
4000000	1708.087602	1421.499509	98843.30832	514.0573943	320.1083167	108951.3776

Table A.23 Maximum Latency Heterogeneous c=0.96

Time Slot	P2Q	P2W	UR	JSQ	JSW	RR
200000	56195.9	11474.3	57668.7	137031.7	9865.4	57723
400000	92145.8	14297	100417.1	210666.6	10749.4	102903.5
600000	133505.8	16990.7	147547.5	274300.8	10986.6	146309.1
800000	152508.4	19197.7	191345.4	332496.1	11214.8	188841.9
1000000	180021.5	21365.5	229728.4	385973.9	11379.6	230617.2
1200000	195518.1	23703.5	265854.2	422004.2	11506	269323.5
1400000	210747.6	25653	307515.7	426216.9	11668.5	312507.3
1600000	242825.8	26814.9	344906.8	436314.9	11668.5	354552.7
1800000	260430.5	29563.8	382962.9	445603.7	11668.5	394889.4
2000000	320273.1	31192.8	425010.8	445603.7	11764.5	437047.6
2200000	348641.6	33990.3	463009	450975.4	11764.5	482511.2
2400000	368368.9	36503.4	500271.4	455129.7	11764.5	518769.4
2600000	388235.8	38597.4	538421	474875.8	11839.2	560212.6
2800000	399378.9	41430.6	575879.1	493491.2	11902.1	598785.6
3000000	440881.9	43860.7	611799.9	510589.3	11902.1	639596.3
3200000	489456.6	45779.1	648153.3	518870.7	11976.9	683812.3
3400000	536648.9	48386.6	692268.3	518870.7	11976.9	717737.4
3600000	558133.5	50063.2	734789.8	526027.4	11976.9	756177.4
3800000	563228.2	52535.1	771162.8	526027.4	11976.9	795227.7
4000000	585235.6	54781.6	807038.7	526027.4	12032.3	839173.6

Table A.24 Average Latency Heterogeneous c=0.96

Time Slot	P2Q	P2W	UR	JSQ	JSW	RR
200000	2713.064663	2390.304571	11188.2293	1432.220417	657.8644125	11993.1314
400000	3596.166251	3169.421925	21540.51384	1546.741786	658.2053132	23375.81015
600000	4289.641575	3801.18306	31887.88184	1609.411939	660.1460003	34731.88568
800000	4930.876904	4340.679263	42243.88902	1647.311248	661.6213682	46101.67973
1000000	5524.267782	4855.632921	52591.74907	1680.065077	663.5305962	57463.8565
1200000	6112.733151	5355.547646	62978.76663	1705.098446	663.464571	68855.94094
1400000	6676.993544	5838.838536	73350.97506	1709.255648	663.0566857	80235.49508
1600000	7221.604355	6306.354438	83707.49211	1702.498145	662.447257	91601.98156
1800000	7741.596157	6769.264087	94039.41457	1697.872558	662.6172816	102955.7245
2000000	8242.724385	7231.115649	104357.2796	1694.693522	663.2680134	114306.5953
2200000	8752.821008	7709.947177	114677.0214	1727.301698	663.7588941	125656.1799
2400000	9249.98749	8197.108264	125007.8223	1745.887482	664.1185925	137015.7267
2600000	9745.054765	8697.311092	135338.7385	1755.917421	664.5531885	148369.3765
2800000	10248.19661	9210.605606	145677.395	1766.297284	665.0749099	159723.2791
3000000	10751.39994	9725.711599	156046.6601	1774.040191	664.9437268	171103.9695
3200000	11257.53577	10245.78733	166411.8676	1784.847679	665.5338885	182478.3754
3400000	11780.34382	10770.32584	176779.8215	1792.974804	665.5814109	193850.9592
3600000	12287.06403	11276.76894	187145.8776	1784.710471	665.0612589	205215.2479
3800000	12782.36876	11773.47455	197516.7444	1783.332384	665.2350861	216584.4535
4000000	13274.88662	12267.95935	207889.4686	1776.944985	664.9313038	227960.1596

Table A.25 Maximum Queue Heterogeneous c=0.96

P2Q	P2W	UR	JSQ	JSW	RR
2197.5	1928.6	11446.5	1208.9	511.9	12373.7
2807.7	2481.1	22113.4	1313.3	517	24114.3
3508.9	3079.6	33047.5	1432.3	533.6	36112.5
4113.4	3585.9	43769.6	1542.8	537.9	47980
4734.4	4095.6	54587.9	1554.5	537.9	59859.9
5292.6	4585.5	65540.8	1605.5	543.4	71770.1
5814.9	5068	76365	1619.6	546.7	83560.9
6310.1	5447	86966.7	1654.4	552.9	95345.3
6728.9	5943.4	97590.1	1682.2	558.8	107051.6
7227	6427.7	108312.3	1761.2	571.8	118954.2
7861	7055.8	119258.5	1818.7	577.4	130804.8
8262.7	7550.3	130114.4	1821.9	577.7	142680.2
8797.8	8255.4	140728.7	1836.8	577.7	154470.2
9449.5	8840.9	151745.8	1886.8	579.8	166386
9905.5	9395	162474.8	1886.8	581.4	178168.4
10523.7	9971.1	173435.8	1889.3	581.5	190190.9
11183.7	10497.5	184284.1	1927.9	583	201894.4
11544.7	10858.1	195148.2	1927.9	591.9	213823.4
12033.3	11367.7	205929.2	1927.9	591.9	225763.8
12448	11751.5	216508.5	1927.9	591.9	237449.9

Table A.26 Average Queue Heterogeneous c=0.96

Time Slot	P2Q	P2W	UR	JSQ	JSW	RR
200000	1411.853344	1243.806302	5824.433589	744.9940135	341.6768615	6243.563569
400000	1870.73752	1648.663656	11209.17289	804.1994093	341.6882595	12164.24088
600000	2231.540166	1977.425698	16593.36194	836.7420393	342.6818772	18073.31277
800000	2565.434004	2258.350435	21984.34964	856.5020915	343.4766755	23992.03305
1000000	2874.840409	2526.846351	27375.15378	873.7247594	344.5384585	29911.22181
1200000	3181.083618	2787.006176	32781.0338	886.7435963	344.4939912	35840.20545
1400000	3474.905554	3038.677679	38180.7699	888.9470557	344.2887894	41764.36782
1600000	3757.75297	3281.481572	43563.80248	885.2569632	343.9093475	47672.39538
1800000	4028.300873	3522.342695	48939.42638	882.8224252	343.987262	53579.6608
2000000	4288.622518	3762.251839	54303.63627	881.0548821	344.2913232	59480.96209
2200000	4554.100203	4011.448808	59674.14141	898.0210465	344.5493726	65387.40127
2400000	4813.115377	4265.2307	65053.71941	907.7420899	344.7561848	71302.65796
2600000	5070.704588	4525.485796	70430.31334	912.9595657	344.9841914	77211.52273
2800000	5332.922302	4792.933363	75815.99294	918.4315714	345.2804899	83126.05075
3000000	5594.606594	5060.798569	81210.21016	922.4381313	345.2022333	89046.42392
3200000	5858.00096	5331.453129	86603.70133	928.064553	345.5061859	94965.04789
3400000	6129.943481	5604.31486	91996.60912	932.2727706	345.5203978	100880.5119
3600000	6393.488899	5867.730738	97389.25847	927.9358441	345.2422272	106792.4663
3800000	6651.354999	6126.315581	102788.5667	927.2329309	345.34054	112711.5423
4000000	6907.482981	6383.476255	108183.0333	923.8788909	345.1709518	118627.6164