Analysing and Quantifying the Influence of System Parameters on Virtual Machine Co-residency in Public Clouds

Abdulaziz Alabdulhafez

A thesis submitted in partial fulfilment of the
degree of Doctor of Philosophy
in Computing Science

School of Computing Science
Newcastle University
September 2015

Abstract

Public Infrastructure-as-a-Service (IaaS) cloud promises significant efficiency to businesses and organisations. This efficiency is possible by allowing "co-residency" where Virtual Machines (VMs) that belong to multiple users share the same physical infrastructure. With co-residency being inevitable in public IaaS clouds, malicious users can leverage information leakage via side channels to launch several powerful attacks on honest co-resident VMs.

Because co-residency is a necessary first step to launching side channel attacks, this motivates this thesis to look into understanding the co-residency probability (i.e. the probability that a given VM receives a co-resident VM). This thesis aims to analyse and quantify the influence of cloud parameters (such as the number of hosts and users) on the co-residency probability in four commonly used Placement Algorithms (*PA*s). These *PA*s are First Fit, Next Fit, Power Save and Random. This analysis then helps to identify the cloud parameters' settings that reduce the co-residency probability in four *PA*s. Because there are many cloud parameters and parameters' settings to consider, this forms the main challenge in this thesis. In order to overcome this challenge, fractional factorial design is used to reduce the number of required experiments to analyse and quantify the parameters' influence in various settings.

This thesis takes a quantitative experimental simulation and analytical prediction approach to achieve its aim. Using a purpose-built VM Co-residency simulator, (i) the most influential cloud parameters affecting co-residency probability in four *PA*s have been identified. Identifying the most influential parameters has helped to (ii) explore the best settings of these parameters that reduce the co-residency probability under the four *PA*s. Finally, analytical estimation, with the coexistence of different populations of attackers, has been derived to (iii) find the probability that a new co-residing VM belongs to an attacker.

This thesis identifies the number of hosts to be the most influential cloud parameters on the co-residency probability in the four *PA*s. Also, this thesis presents evidence that VMs hosted in IaaS clouds that use Next Fit or Random are more resilient against receiving co-resident VMs compared to when First Fit or Power Save are used. Further, VMs in IaaS clouds with a higher number of hosts are less likely to exhibit co-residency.

This thesis generates new insights into the potential of co-residency reduction to reduce the attack surface for side channel attacks. The outcome of this thesis is a plausible blueprint for IaaS cloud providers to consider the influence on the co-residency probability as an important selection factor for cloud settings and *PA*s.

Acknowledgements

I would like to thank everyone who has offered me support and advice during my Ph.D., especially my supervisor Dr. Paul Ezhilchelvan, Professor Isi Mitrani and my second supervisor Dr. Feng Hao.

Declaration

All work contained within this thesis represents the original contribution of the author. This thesis includes work that has been published in peer-reviewed publications. These publications are as follows:

- A. Alabdulhafez and P. Ezhilchelvan, “Experimenting on Virtual Machines Co-residency in the Cloud,” in *Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC ’14*, 2014, pp. 363–366.
- A. Alabdulhafez and P. Ezhilchelvan, “Quantifying the Risk of Multi-tenancy in the Cloud,” in *Proceedings of the 6th York Doctoral Symposium on Computer Science and Electronics - YDS ’13*, 2013, pp. 73–78.
- A. Alabdulhafez and P. Ezhilchelvan, “Analyzing the Success Rate of Virtual Machines Co-residency in the Cloud,” in *Proceedings of the 6th Saudi Scientific International Conference (SIC)*, 2012, pp. 164–168.

Content

List of Figures

List of Tables

Glossary of Notation and Abbreviations

IaaS	Infrastructure-as-a-Service
SaaS	Software-as-a-Service
PaaS	Platform-as-a-Service
VM	Virtual Machine
PA	VM Placement Algorithm
VMC	Virtual Machine Co-residency simulator
γ	VM Requests Arrival Rate
α	Attackers VM Requests Ratio
k	Total Number of Co-residency Hits
$n_{malicious}$	The total number of placed malicious VMs in the cloud
n_{hit}	The total number of VMs that experienced at least one hit in the cloud
$n_{hit\ by\ malicious}$	The total number of VMs that experienced at least one malicious hit
LT	VM Lifetime
CCP	Co-residency Coverage Probability
HFL	Hit-free Lifetime Ratio
CV	Co-residency Vacancy
CA	Co-residency Activity
MCP	Malicious Co-residency Probability
AFL	Attacker-free Lifetime Ratio
ANOVA	Analysis of variance test
r-value	The sample Pearson correlation coefficient

Chapter 1
Introduction

This chapter presents the context and motivation for this thesis. Then, the research approach, questions and statement are stated. Finally, the main contributions of the research and an overview of the thesis structure are presented.

1.1 Context and Motivations

Recent advances in cloud computing encourage businesses and organisations to host services and applications in third-party public clouds. A recent study on the cloud usage [63] showed that approximately 30% of IT organizations use public clouds such as Microsoft's Azure [57] and Amazon's EC2 [4]. These clouds provide Infrastructure-as-a-Service (IaaS) allowing individuals and organizations to host services on-demand, and paying just for what they have consumed. Businesses and governmental bodies may even use applications hosted in the cloud to access highly sensitive internal records. However, this rapid increase in the adoption rate of public IaaS cloud has resulted in the need for increased security.

To achieve maximum utilization of their physical infrastructure, IaaS cloud providers allow multi-tenancy ending with co-residency. Multi-tenancy is where virtualization is used to enable multiple users (tenants) to share the same physical host. Co-residency is multiple co-residing Virtual Machines (VMs) belonging to different users being hosted by the same physical host.

Enabling co-residency can be cost-effective for IaaS cloud providers. However, co-residency has been shown to be one of the effective avenues for launching several easy-to-implement but powerful attacks on honest (i.e. non-attacker) co-resident VMs using side channels.

1.1.1 Co-residency and Side Channel

A side channel is a form of information leakage that arises as a result of sharing physical resources with other users. For example, the sharing of the CPU and memory caches, that has been shown by [13], [79] and many others to be a vulnerability that can be compromised to bypass VMs isolation. Side channel attacks in multi-tenant environments have been demonstrated by many researchers (see Section 2.3.2) to threaten the security of VMs, particularly in public IaaS clouds.

Researchers have introduced an increasing number of low-cost side channel attacks that can be launched after achieving co-residency. Using Amazon EC2 as a case study, Ristenpart *et al*'s [79] pioneer research demonstrated that side channel attacks targeting specific VMs are possible. They proved this after they successfully placed malicious VMs to become co-resident with up to 40% of target VMs. Such action can have huge negative consequences for the honest co-resident VMs that belong to businesses and organizations. An attacker may be able to measure the host CPU cache usage to determine, for instance, how busy the co-resident VM is, but this might be a smaller concern of the co-resident VM's owner. More seriously, an attacker can use side channels to degrade co-resident VMs' performance by more than 80% [92]. Alternatively, worse by running Denial of Service on co-residing VMs to block the cloud customers from accessing the compromised cloud services [53].
Even more seriously, a co-resident attacker may be able to steal decryption and secret keys, such as ElGamal decryption keys [103], RSA [75] and AES [70] secret keys. Then executing malicious code in the host operating system [94]. Such action can result in breaches of privacy of the VMs running in public IaaS clouds, allowing co-resident attackers to eavesdrop on communications and steal sensitive data and make it public.
The aforementioned security threat brought about by side channel attacks is amplified by the fact that attackers can run their malicious VMs in the cloud legitimately as long as they have access to the Internet and a payment method. Worse, an increasing amount of research in recent years has introduced new side channels [101]. Consequently, anyone who has access to the Internet, from any location, can attempt to co-reside and attack honest VMs, using any side channel they choose.

1.1.2 Side Channel Attacks Countermeasures

A public IaaS cloud uses a VM Placement Algorithm *(PA*) that controls where each new VM is placed, possibly to become co-resident with other VMs sharing the same host. Common practices to secure such shared environments usually include relying on virtualisation to ensure strong isolation between co-resident VMs so that they become unable to interfere with each other [9]. However, virtualized isolation that completely prevents side channels has been proven to be difficult to achieve. The following countermeasures address side channel attacks at the cloud provider side, the cloud user side and the hardware/software vendor's side respectively:

(1) Physical isolation enforcement: it can be argued that one pragmatic solution to mitigate side channel attacks is to disable co-residency completely. Ristenpart *et al* [79] suggested that cloud customers (businesses or governmental bodies) may consider running their VMs in physical isolation from other VMs. Following this suggestion, the Amazon EC2 cloud allows users to run dedicated VMs [28], ensuring that VMs belonging to each user do not share the same physical hardware with any other cloud users' VMs. Although this service can effectively mitigate various side channels that exist in the shared hardware, significant price premiums are required for cloud users to use this service. It is estimated that it is 6.12 times more costly to run dedicated VMs compared to using regular VMs in Amazon EC2 [97]. This significant extra cost of the dedicated VMs diminishes its attractiveness, coupled with the fact that enabling co-residency is a definite choice of IaaS cloud providers due to its economic efficiency.

One of the options left for protecting VMs from side channel attacks is to allow only other "trusted" VMs to become co-resident. If untrusted VMs become co-resident, then relocate the user's VM to another host [11]. Trusted VMs, in this case, may include VMs that are self-owned or other trustworthy VMs. However, this requires enabling cloud users to audit and verify the cloud provider's adherence to this policy, where the work of [102] has introduced a promising tool to help with this issue.

2) Allowing the cloud user to specify where to place his VMs: Although this countermeasure is relatively straightforward, it does not solve the problem completely. In fact, it only shifts the liability to the user instead of the cloud provider without trying to eliminate the side channel or the side channel attacks.

3) Preventing side channel vulnerabilities: This can be achieved via reducing the information that can be leaked by new cache hardware designs or by applying various blinding techniques. For example, using non-deterministic caches and cryptographic implementation of timing-resistant caches (see Section 2.3.3.3). However, [79] concluded that countermeasures that rely on preventing side channels vulnerabilities suffer from two major drawbacks. First, they are typically (a) impractical, for instance, incurring high overheads or requiring nonstandard hardware or they are (b) application-specific or hardware-specific. Second, these countermeasures do not,

ultimately, guarantee that all possible side channels have been anticipated and disabled, especially in the light of the increasing number of research in recent years that introduce new side channels.

Despite the efforts being paid to VMs safeguarding against existing side channels, there remains a continuous potential risk of data leakage by new side channels that are yet to be discovered. Therefore, this opens an interesting research area to find an alternative approach to reduce the attack surface for side channel attacks, particularly one that does not rely on VMs physical isolation or side channels prevention.

1.2 Thesis Aim and Approach

Because co-residency is a necessary first step to launching side channel attacks, this motivates this thesis to look into understanding the co-residency probability. The co-residency probability is defined as the probability that a given VM receives a co-resident VM (i.e. honest or malicious VM) during its lifetime.

The main aim of this thesis is to quantify and analyse the influence of cloud parameters (such as the number of hosts and users) on the co-residency probability under four commonly used *PA*s. These *PA*s are First Fit, Next Fit, Power Save and Random. This action then leads to identifying the influential parameters' settings that reduce the co-residency probability in each *PA*. Reducing the attack surface for side channel attacks is one outcome of reducing the co-residency probability.

This thesis achieves its aim through quantitative experimental simulation and analytical prediction. This approach consists of four main steps:

(1) **Characterizing** the co-residency occurrence behavior in IaaS clouds using co-residency metrics, followed by

(2) **Identifying** the most influential cloud parameters (such as the number of hosts, clusters and users) affecting co-residency probability in four *PA*s. To do so, the influence of all relevant cloud parameters is quantified.

(3) **Simulation** experimentation to find the best settings of the most influential parameters that reduce the co-residency probability under each *PA*.

(4) **Analytical** estimation, with the coexistence of different populations of attacker VMs, to find the probability that a new co-residing VM belongs to an attacker. These estimates help in identifying the best *PA*s that reduce the probability above.

Each of the above steps is addressed in a separate chapter that details how the step will be executed.

The scope of this thesis is limited to public IaaS clouds only because the higher risk of side channels is usually associated with publicly accessible IaaS clouds where an attacker is able to fully control malicious VMs to attack co-resident VMs [79].

1.3 Research Hypotheses and Questions

The following **two hypotheses** are proposed:

1. For a given *PA*, cloud parameters, such as the number of hosts and users, do not have the same influence on the co-residency probability in IaaS clouds.
2. For a given VM, there is a non-zero probability that a new co-residing VM belongs to an attacker for any of the four *PA*s considered.

Based on the aim above and research hypotheses, the following research questions are explored:

1- How to characterise the co-residency occurrence behaviour in IaaS clouds?

To experiment on co-residency in this thesis, the co-residency occurrence behaviour is characterised using four quantitative metrics (referred to as the co-residency metrics). Some of these characteristics include how likely a given VM *u* will be co-resided by another VM *v*, as well as how long this co-residency takes to occur. These metrics play a significant role in answering the remaining research questions, and should also be useful to further research on co-residency in IaaS clouds.

2- For a given *PA*, what are the most influential cloud parameters affecting co-residency probability?

Modelling large-scale and dynamic environments, such as IaaS clouds, involves several parameters; some of them could exercise higher influence on the co-residency probability than others. For simplicity, this thesis focuses on the cloud parameters that have the most influence on the co-residency metrics. An Influence Evaluation Strategy is proposed to quantify the influence on the co-residency metrics across a variety of likely cloud parameters' settings under four *PA*s. These *PA*s are First Fit, Next Fit, Power Save and Random (described in detail in Appendix A). The strategy uses fractional factorial designs [15] to design the experiments and applies Analysis Of Variance (ANOVA) tests

to identify the most influential parameters and parameters interactions on the co-residency metrics.

3- For a given *PA*, which parameter settings reduce the co-residency probability?

The most influential parameters are used in controlled experiments to estimate the co-residency metrics using a wide range of settings under four *PA*s to allow:

(i) Identifying the best parameters' settings where a given *PA* can reduce the co-residency probability.

(ii) Identifying any situations where selecting parameters' settings in a given *PA* would not be able to reduce the co-residency probability.

(iii) Identifying the best *PA*s, regardless of the parameters' settings, that reduce the co-residency probability.

4- For a given VM, what is the probability that a new co-residing VM belongs to an attacker?

The risk of side channel attacks is magnified enormously if an honest VM is co-resided by an attacker VM. Therefore, this research question investigates reducing the malicious co-residency probability (i.e. the probability that the next co-residing VM belongs to an attacker). Two approximate analytical estimates are derived to estimate the malicious co-residency probability with the coexistence of different populations of attackers. These estimates also help IaaS cloud providers to find the best *PA*s that can hinder attackers from easily achieving malicious co-residency.

Each of the above research questions is addressed in a separate chapter that describes the approach used to answer the question, followed by a discussion of the important findings. The previous research questions can be summarized in the following thesis statement.

1.4 Thesis Statement

Co-residency is a necessary first step to launching several side channel attacks that have been shown to threaten the security of users' VMs in public IaaS clouds. Therefore, this thesis looks into understanding the co-residency probability. This thesis aims to analyse and quantify the influence of cloud parameters (such as the number of hosts and users) on the co-residency probability in four commonly used *PA*s. These *PA*s are First Fit, Next Fit, Power Save and

Random. This analysis helps to identify the appropriate cloud parameters' settings that reduce the co-residency probability in four *PA*s. Quantitative experimental simulation and analytical prediction approach are used to achieve the aim of this thesis.

1.5 Challenges

Studying co-residency occurrences behavior in the large, non-transparent and diverse IaaS clouds can become a very challenging task that requires an efficient testing methodology (i.e. testbed). Such a testbed must support experimentation under different scenarios and settings and, most importantly, a number of *PA*s. There is no single or best testbed that supports experimenting with many parameters that describe IaaS cloud architecture, functional and non-functional requirements. Based on the discussion in Appendix F, there are three testbeds that can be used for this thesis's experiments:

1- **Public IaaS clouds**
2- **Private IaaS clouds**
3- **Simulators**

The above testbeds are evaluated in Appendix F for their suitability to conduct the thesis experiments, highlighting the limitations and advantages of each testbed. The choice of the testbed follows from the research aim that requires exploring the influence of various parameters on the co-residency metrics using different settings of IaaS clouds under four *PA*s. This evaluation nominates simulation to be a flexible and cost-effective testbed [1]. Therefore, simulation experimentation is adopted and the VM Co-residency simulator *VMC* was implemented and used as a testbed in this thesis. The *VMC* simulates the thesis's system and attack models (Chapter 3) and uses the co-residency metrics to estimate different probabilities related to co-residency.

Another challenge is that there are many cloud parameters and parameters' settings to be included in limited resources experiments in this thesis. Therefore, fractional factorial design is applied that helps to construct a reduced and balanced experiment. Fractional factorial experiments are usually used to measure simultaneously the effects of many parameters on a product or process in a cost-effective way using minimal experimental runs [33].

Also, estimating the malicious co-residency probability under various attackers ratios (i.e. the proportion of attacker VMs to total VMs) introduces another challenge. Exploring all likely attackers ratios, using simulation, is an attempt that resource and time limitations did not

allow. To overcome this challenge, analytical estimates are derived that take into account the attacker ratio (see Chapter 6).

1.6 Contributions

In the course of responding to the research questions, each chapter makes contributions to the field of VMs security in IaaS cloud. The main contributions of this thesis are as follows:

First, defining four quantitative metrics to statistically characterise the probability of co-residency occurrences. Some of these characteristics include how likely a given VM *u* will be co-resided by another VM *v*, as well as how long does this co-residency take to occur. While there has been work done in the area of co-residency, to the best of one's knowledge this thesis is the first to characterise co-residency probability using quantitative metrics. These co-residency metrics proved to be very useful in answering the research questions in this thesis, and should also be useful for future research related to co-residency in IaaS clouds.

Second, quantifying the influence of cloud parameters on the co-residency probability under four *PA*s (First Fit, Next Fit, Power Save and Random). This action leads to identifying the most influential parameters and parameter interactions on VM co-residency. A novel Influence Evaluation Strategy is proposed for assisting researchers to identify the most influential parameters on the co-residency metrics in large-scale, dynamic IaaS clouds. This strategy can be applied to assess the effect of varying multiple cloud parameters on the co-residency metrics such as varying the rate at which VM requests are generated, using a different number of hosts and others. The Influence Evaluation Strategy identified the number of hosts to be the most influential cloud parameter on the co-residency probability in four *PA*s.

Third, identifying the appropriate cloud settings in four *PA*s that reduce the co-residency probability. Reducing the co-residency probability aims to reduce the attack surface for side channel attacks. In order to identify the appropriate cloud settings, simulation experiments explored how the most influential parameters' settings in four *PA*s could positively and negatively affect the co-residency metrics. The simulation experiments were conducted under a wide range of likely settings for publicly accessible IaaS clouds. The experiments present evidence that VMs hosted in IaaS clouds that use Next Fit or Random are more resilient against receiving co-resident VMs compared to when First Fit or Power Save are used. Further, VMs in

IaaS clouds with a higher number of hosts are less likely to exhibit co-residency. The outcome of this thesis is a plausible blueprint for IaaS cloud providers to consider the influence on the co-residency probability as an important selection factor for cloud settings and *PA*s.
While an increasing number of literatures have compared *PA*s in several aspects such as cost reduction [37], [48], [49] and performance and energy consumptions [40], [55], [58], [99], this thesis is the first to compare *PA*s in terms of how they affect the co-residency probability.

The **fourth** contribution is deriving analytical estimates of the co-residency probability that take into account the number of attacker VMs in the IaaS cloud. These estimates can be used by anyone to determine analytically, with the coexistence of a given number of attacking VMs, the best *PA*s that reduce the malicious co-residency probability. That is to say that a new co-residing VM belongs to an attacker. Comparing First Fit, Next Fit, Power Save and Random *PA*s, the analytical estimation shows that the malicious co-residency probability varies widely from one *PA* to another. The analytical estimation shows that the right choice of *PA*s can reduce the likelihood of being co-resided by attackers' VMs. In addition, these estimates are proved to compare well with the experimental estimates (i.e. using the *VMC* simulator). Therefore, the derived analytical estimates should become very useful for IaaS cloud providers and users for estimating the malicious co-residency probability in various IaaS cloud's settings, *PA*s and number of attacker VMs.

Fifth, introducing a new VM Co-residency (*VMC*) simulator that allows modelling of co-residency behaviour using various cloud parameters' settings and *PA*s. The *VMC* can be used as an experimentation tool for assessing the influence of cloud parameters on the co-residency probability. The *VMC* simulator also allows the exploration of the appropriate parameter settings that reduce the co-residency probability in a given *PA*. The *VMC* has been used successfully as a testbed in this thesis and should also be useful in advancing future research related to VMs co-residency in IaaS clouds.

1.7 Thesis Structure

The remainder of this thesis consists of the following chapters:

- Chapter 2 provides a background on cloud computing and related literature. Further, the chapter considers the key issues of side channels in public IaaS clouds and then explores common side channel attacks that can be launched against co-resident VMs. Available

countermeasures against side channels, including physical isolation and side channel vulnerabilities prevention, are shown to have some drawbacks. Therefore, this thesis looks into understanding the co-residency probability so that the best cloud parameters' settings that reduce the co-residency probability are identified in four *PA*s. Co-residency reduction can contribute to reducing the attack surface for side channel attacks.

- Chapter 3 begins by describing how the IaaS cloud is modelled in this thesis. Next, the co-residency metrics are defined to address the first research question. The co-residency metrics are quantitative measurements that characterise different probabilities related to co-residency occurrences in IaaS clouds. These co-residency metrics are used to identify the most influential parameters on co-residency (i.e. the second research question), as well as to find the best parameter settings in each *PA* that reduce the co-residency probability (i.e. the third research question). In addition, the co-residency metrics are used to derive analytical estimates of probabilities related to malicious co-residency (i.e. the fourth research question).

- Using the *VMC* simulator as a testbed, Chapter 4 defines the Influence Evaluation Strategy and applies it to answer the second research question on what cloud parameters influence the co-residency metrics the most. Under First Fit, Next Fit, Power Save and Random *PA*s, the strategy quantifies the influence of cloud parameters on the co-residency metrics then identifies the most influential parameters and 2-parameter interactions. In addition, the strategy provides useful insights that are used to compare the *PA*s in terms of their impact on the co-residency metrics. Further, the results presented in this chapter examine the first hypothesis put forward in Section 1.3.

- Chapter 5 answers the third research question of "the parameter settings in a given *PA* that reduce the co-residency probability". The *VMC* simulator is used to estimate the co-residency metrics under four *PA*s using different settings of the most influential parameter. Pearson's correlation analysis [14] is applied to study the correlation between parameters and the co-residency metrics. Then, the best parameter settings in four *PA*s that effectively reduce the co-residency probability are identified.

- Chapter 6 considers the probability that, for a given VM, the next co-residing VM belongs to an attacker (i.e. the fourth research question). Two analytical estimates of the malicious co-residency probability are derived and calculated. These analytical estimates are used to find, with the coexistence of different populations of attacking VMs, the probability that a new co-residing VM belongs to an attacker. The *VMC* simulator is used to validate these analytical approximations using the four *PA*s. The outcome of this validation shows an agreement between the analytical estimates and the simulation estimates across the four *PA*s.

- Chapter 7 draws conclusions as to analysing and quantifying the influence of cloud parameters on the co-residency probability in public clouds. In addition, this chapter summarizes how the co-residency probability has been reduced through identifying the appropriate parameters' settings in each *PA*. Finally, possible future work is discussed.

Chapter 2
Background and Related Work

2.1 Introduction

This chapter begins by looking into cloud services and models; highlighting the crucial role that virtualization plays in making the cloud a cost-effective solution for businesses and organisations.

In Section 2.2, virtualisation is shown to have brought a security threat to multi-tenant public IaaS clouds, where VMs belonging to different users share the same physical host (i.e. co-residency).

Section 2.3 surveys the related work on co-residency, presenting different methods for achieving co-residency with a target VM and the techniques used to detect successful co-residency.

Section 2.3.2 looks at co-residency as an attack avenue. Several easy to implement, yet harmful, side channel attacks that can be launched against co-resident VMs are discussed and shown to bring a significant threat to VMs security in the cloud.

Available countermeasures against side channels, including physical isolation and side channel vulnerabilities prevention, are shown to have some drawbacks in Section 2.3.3.

Because co-residency is a necessary first step to launching side channel attacks, understanding the co-residency probability (defined in Section 1.2) is identified as an interesting research gap in Section 2.3.3.4. Consequently, this thesis explores reducing the co-residency probability through the right choice of the cloud parameters' settings in four *PA*s. Reducing the co-residency probability can contribute to reducing the attack surface for side channel attacks.

2.2 Cloud Computing

Recent advances in cloud computing encourage businesses and organisations to host services and applications in third-party public clouds. Since 2007, the term cloud has become an overused buzzword in the IT industry. Many definitions of cloud computing have been suggested from different application aspects. However, there is no agreed consensus definition for cloud computing. The U.S. National Institute of Standards and Technology, NIST, [56] provides an interesting definition that specifies essential characteristics of cloud

computing and delivery and deployment models as well. This definition is quoted as follows: "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."
The following sections outline three service models, four deployment models as suggested in [56] and then look into technical aspects of cloud computing related to this thesis.

2.2.1 Cloud Service Models

Based on the delivery mechanism, cloud services are categorized into three major models: software, platform and infrastructure.

2.2.1.1 Software as a Service (SaaS)

In this model, the cloud user is given the ability to use certain applications hosted in a cloud infrastructure. These applications are normally accessible through a web browser. What is unique about this model is that cloud users are unable to control or manage the application configurations or the underlying cloud physical infrastructure including network, servers and operating systems.

2.2.1.2 Platform as a Service (PaaS)

Unlike the SaaS model, the users in this model can deploy applications, services and tools onto the cloud infrastructure given that the provider supports them. Although the users usually have control over the deployed application's configurations and settings, they are still unable to manage or control the underlying cloud infrastructure including network, servers and operating systems.

2.2.1.3 Infrastructure as a Service (IaaS)

In this model, more capabilities are given to the cloud users. Among these capabilities is the provisioning of essential computing resources including processing, networks, and storage. The cloud user can deploy and run any operating systems and applications using dedicated and self-controlled VMs that are allocated into virtualised hosts using VM Placement Algorithms *(PA*s).

Based on a comparison between the previous cloud service models, a cloud service is assumed to become more vulnerable as more capabilities are given to the users.

2.2.2 Cloud Deployment Models

Based on the usage scopes, cloud infrastructure can be deployed in different fashions. The following are the four primary cloud deployment models:

2.2.2.1 Private Cloud

The cloud infrastructure in this model can be used exclusively by a single organization. The entire infrastructure may be self-owned and managed by the organization or outsourced to a third party irrespective of whether the cloud infrastructure is hosted on or off the premises.

2.2.2.2 Community Cloud

The cloud infrastructure in this model can be used exclusively by a specific community of users from organizations that share specific concerns such as security and compliance requirements. The entire infrastructure may be owned and managed by one or more of the community organizations or outsourced to a third party irrespective of whether the cloud infrastructure is hosted on or off any of the organisations' premises.

2.2.2.3 Public Cloud

The cloud infrastructure in this deployment model is open for public use. The entire infrastructure is owned and managed by third-party providers and hosted on their premises. What is unique about this deployment model is that it is publicly accessible, and any user with access to the Internet and a payment method can legitimately use the cloud.

2.2.2.4 Hybrid Cloud

The cloud infrastructure in this deployment model is a combination of two or more cloud deployment models (private, community, or public).

2.2.3 Technical Aspects

The following are essential technical characteristics that are used in the cloud to support specific functional and economical requirements.

2.2.3.1 Virtualization

Virtualization is an essential characteristic of cloud computing and helps to deliver the value of cloud computing. Virtualization is a technology that separates physical infrastructures to create various "virtual" dedicated resources. Virtualization allows the running of multiple operating systems and applications, using VMs, on the same physical host concurrently [63].

2.2.3.2 Multi-tenancy

Virtualization can effectively maximize the utilization of physical resources if multi-tenancy is enabled, which is sharing of physical resources between multiple users (i.e. co-residency). Moreover, multi-tenancy brings an important economic benefit to cloud providers through sharing operating expenses between users in order to provide cost-effective cloud services.

2.2.3.3 Security

Although cloud computing has its advantages, it also comes with new threats. Before the cloud era had begun, data and applications were only deployed in the users' own infrastructure inside their premises, and, therefore, the users' data could be under their control and supervision and physically secured. On the other hand, exporting applications and data to a third-party cloud brings several threats to confidentiality, integrity and availability [78].

Among all the possible threats in the cloud environment, this thesis focuses on a threat brought by multi-tenancy (i.e. co-residency). Although enabling multi-tenancy can be cost-effective for public IaaS cloud providers, the following sections show that co-residency is one successful avenue for launching several easy-to-implement and powerful attacks on honest co-resident VMs using side channels.

2.3 Related Literature

As mentioned in Section 1.2, the scope of this thesis is limited to public IaaS clouds only. This is because the higher risk of side channels is usually associated with publicly accessible IaaS clouds where an attacker is able to fully control malicious VMs to attack co-resident VMs.

Public cloud services provide IaaS allowing individuals and organizations to run and control VMs, and paying only for what they have consumed. Examples of public cloud services include Microsoft's Azure [57] and Amazon's EC2 [4]. In order to achieve maximum utilization of their physical infrastructure, cloud providers allow multi-tenancy [84], ending with many co-resident VMs sharing the same underlying physical infrastructure (e.g. host CPUs and memory). Common practices to secure such shared environments usually include ensuring strong isolation between co-resident VMs so that they become unable to interfere with each other. In addition, each VM should be unaware of other VMs running in the same physical host [9]. However, co-residency can lead to a risky situation where an attacker VM ends up residing on the same physical host with victim VM(s). An attacker's co-residency with a victim VM gives an opportunity to launch several possible harmful attacks using side channels [103]. The next sections discuss related literature on co-residency, followed by a list of various types of side channel attacks and the available countermeasures, leading to identifying the research gap addressed by this thesis.

2.3.1 Co-residency

A number of research publications discuss different aspects of co-residency, where preliminary work on identifying co-residency as an existing threat in public IaaS clouds was undertaken by [79]. Their work demonstrates that the co-residency imposes a risk on sensitive services and data hosted in third-party public IaaS clouds. Their study was conducted in order to answer the following concrete questions:

1- Is it possible to determine where in the cloud a VM is hosted (located)?
2- Is it possible to determine whether two VMs are co-resident on the same physical host?
3- Is it possible to place a malicious VM to be co-resident with a victim VM?
4- Is it possible for an attacker to launch attacks that use side channels against co-resident VMs?

Using Amazon EC2 as a case study, researchers have shown that the answer to these four questions is "yes". They demonstrated this by mapping Amazon EC2's internal cloud infrastructure in order to locate where specific targeted VMs are likely to reside. Then, malicious VMs are launched until one or more of these VMs become co-resident with the targeted VMs. In addition, several scenarios, where an attacker can launch variant side channel attacks to collect sensitive information from co-resident VMs, were demonstrated.

The authors claim that the method that they applied to achieve co-residency helped to co-reside with up to 40% of the target VMs.

The methods to achieve co-residency discussed next are related to the work done in the Amazon EC2 by [79]. Their method consists of the following three steps:

- Locating victim VMs (cloud cartography),
- Malicious VMs Placement,
- Co-residency detection and then launching side channel attacks against co-resident VMs.

2.3.1.1 Locating Victim VMs (Cloud Cartography)

In order to locate a specific target VM, the EC2 cloud's internal physical infrastructure is mapped using the DNS services to resolve public DNS names to public IPs. Then, available network tools, such as nmap, hping and wget can be used to map each public IP to its private IP equivalent inside the EC2 cloud. This mapping process can provide a better understanding of how the cloud infrastructure is constructed.

2.3.1.2 Co-residing Techniques

Achieving co-residency with a particular victim in Amazon's EC2 requires a good understanding of the cloud infrastructure, how the network addresses are assigned to each VM and the used *PA*. The following co-residing techniques have been applied to achieve targeted co-residency using the Amazon EC2's cloud cartography that was constructed in the previous step:

- **Brute-force**

Using the cloud cartography, it is possible to determine roughly in which zone, in Amazon EC2, the targeted VMs are, allowing the attacker to create as many VMs as possible in the same zone. The authors claim that they achieved an average of 8.4% successful co-residencies with a set of target VMs using brute-force.

- **Placement Locality Abuse**

Using Amazon's EC2, the authors prove that VMs created with small time gaps are more likely to be placed in the same host. The placement locality abuse

technique involves detecting when the victim VMs are launched, then immediately creating malicious VMs with the hope that the *PA* will place them in the same host as the target(s). Abusing the placement locality allowed the malicious VMs to co-reside with 40% of the target VMs.

2.3.1.3 Detecting Co-residency

Co-residency detection through side channels was first exposed by [79]. After placing many malicious VMs in different hosts, the attacker must check whether the target victim VMs and any of the attacker's VMs are co-resident or not. This can be achieved using one of the following co-residency detection checks:

- **Matching Dom0 IP (Xen-specific)**

Dom0 is a special-privileged process created in hosts that run Xen virtualization [21]. One of the Dom0 main jobs is to manage traffic routing between the co-resident VMs. This co-residency detection check uses the network command (`trace route`) to trace network packets that are sent to a victim VM, comparing if the:

First hop = attacker's VM's Dom0 IP address
Last hop = victim VM's Dom0 IP address

The victim VM and the attacker's VM are in the same host (i.e. co-resident) if the victim VM's Dom0 IP address matches the attacker VM's Dom0 IP address. However, this method is specific to hosts that run Xen virtualization. Also, it assumes that the Dom0 process responds to traceroute commands. Bates *et al* [13] state that this co-residency detection check is no longer applicable in Amazon EC2.

- **Network Packet Round Trip Times**

One of the simplest and easiest ways to detect co-residency is by measuring the travel time of network packets sent from an attacker VM to a victim VM and then to compare it with the time the same packet takes to reach other VMs that reside in other hosts. Ristenpart *et al* demonstrated that it was possible to predict the co-

residency of two VMs if the packet round trip time is lower when sent to a co-resident victim VM [79].

- **Non-network Based Co-residency Checks**

  It is possible for the IaaS cloud providers to disable the use of all network tools offered to the cloud users. In this case, other non-network based co-residency checks that use side channels are presented in a number of papers. For instance, [79] demonstrated that an attacker can send a heavy load (e.g. HTTP requests) to the target VM and can observe the CPU load of the attacker VM's host. If the CPU load has increased, then this a sign that the attacker VM shares the same host with the target VM. Other side channel co-residency detection checks include using cache-based side channel attacks [100] and using Active Traffic Analysis [13].

Once an attacker succeeds in achieving and detecting a co-residency with a victim VM, various possible side channel attacks can be launched to collect sensitive information about the victim VM as shown next.

2.3.2 Side Channel Attacks

A side channel is a well-known security threat in multi-tenant systems. With a history that goes back to 1972 [47], the threats of side channels are frequently present in systems where users share physical resources, such as memory, network bandwidth and CPU caches.
A side channel is a form of information leakage that arises as a result of sharing physical resources with other users. For example, sharing of the CPU and memory caches has been shown by [13], [79] and many others to be a vulnerability that can be compromised to bypass VMs isolation.
Side channel attacks in multi-tenant environments have been demonstrated by many researchers to threaten the security of VMs, particularly in public IaaS clouds [8]. Researchers and hackers alike have introduced an increasing number of low-cost side channel attacks that can be launched after achieving co-residency [16], [23], [24], [25]. Using Amazon EC2 as a case study, Ristenpart *et al*'s pioneering research demonstrated that side channel attacks are possible. Ristenpart *et al* proved this possibility after they successfully placed malicious VMs to become co-resident with up to 40% of target VMs [79]. Such actions can have huge negative consequences for the honest co-resident VMs.

Examples of these side channel attacks include:

- **Key leakage** such as extracting RSA [75] and AES [33] secret keys.
  In addition, the researchers in [103] have been able to extract ElGamal decryption keys form a co-resident VM. They demonstrated this attack using the classic Prime+Probe technique presented in [70]. The first step of this technique involves priming the CPU cache (data or instruction) by accessing a certain range of addresses that cause the cache to become full. The attacker then yields the CPU, which in turns allows the victim VM to evict some of the attacker's data or instructions from the cache. Immediately, the attacker preempts the victim and starts probing the cache again by accessing a certain range of addresses that cause the cache to become full. At this stage, the attacker can measure the time taken for each cache access in order to determine which cache lines were replaced by the victim. This action allows the attacker to learn some information about the addresses that have been accessed by the victim. By studying how standard libraries implement the private ElGamal decryption key, the researchers show that monitoring the victim's repeated exponentiations for a few hours allows the attacker to reconstruct the 457-bit private exponent of a 4096-bit modulus.
  Secret keys leakage can result in breaches of privacy of the VMs running in public IaaS clouds, allowing co-resident attackers to eavesdrop on communications and steal sensitive data and make it public.
- **Running Denial of Service (DOS)** on a co-resident VM, blocking the compromised VM's owners and users from access. The researchers in [53] have introduced a new form of DOS attack in IaaS clouds where they map the cloud network topology to identify and starve an uplink bottleneck of a victim VM. The attack requires co-residing with a victim VM as well as allocating other attacker's VMs into enough hosts within the same subnet. Then the attacking VMs are used to flood the uplink to the victim VM with high UDP traffic. The immediate side effect of this attack is starving other important TCP sessions of the victim VM as a result of the TCP congestion avoidance mechanisms.
- **Exploiting a heap buffer overflow** to execute malicious code in the host operating system [94].
- **Determining web traffic rates**. Side channel load measurement can be used to estimate the number of web visitors to a co-resident VM, or even the most frequently

visited pages. This information might be damaging if, for example, the co-resident VM belongs to a corporate competitor. The researchers in [79] were able to estimate the HTTP traffic rates to a co-resident VM using load measurement technique. This technique involves performing cache load measurements while sending different rates of HTTP requests to the victim VM. This action will allow the attacker to correlate between traffic rates to the victim VM and load samples. Although this type of information may sound harmless, it can be used to work out targeted VMs activity patterns and peak trading times for maximum Denial of Service effect.

- **Performance-Degrading attacks** such as the Swiper attack [20] and the Resource-freeing attack that was demonstrated to degrade co-resident VMs' performance by more than 80% [92].
- **Gathering sensitive information via side channels** as demonstrated in Amazon EC2 by [79] and included:
  - Non-network based co-residency detection (see Section 2.3.1.3).
  - Measuring the CPU cache usage of the targeted VMs to determine, for instance, how busy the VMs are.
  - Exploiting the memory bus as a high-bandwidth side channel for data transmission [97].
- **Keystroke timing:** stealing SSH passwords from co-resident VMs as shown by [87] and [36].
- **Several application-specific side channels** that have been reported to allow attacking VMs to exploit co-resident VMs isolation. For instance, attackers were able to steal and leak the VMware ESX hypervisor source code [16]. Because the hypervisor is responsible for controlling the traffic between co-resident VMs, this source code leakage potentially allows the attackers to find ways to eavesdrop on co-resident VMs. Another vulnerability in Xen-based clouds has been reported that allows a guest VM to execute arbitrary commands in the hypervisor [23]. Moreover, a number of integer overflow vulnerabilities have been reported to affect the e2fsprogs packages (these packages contain a number of utilities for ext2 and ext3 file systems in Linux). An attacker can target a VM by –remotely- tricking the VM's owner into opening a malicious file in order to execute arbitrary code with the same permissions as the victim. Worse, an attacker can gain access to other virtualized hosts by exploiting this vulnerability as shown in [24] and [25].

The aforementioned security threats brought about by side channel attacks is amplified by the fact that attackers can run their malicious VMs in the cloud legitimately as long as they have access to the Internet and a payment method. In addition, an increasing amount of research in recent years introduces new side channels [101]. Therefore, anyone who has access to the Internet, from any location, is able to attempt to co-reside and attack honest VMs, using any side channel they choose.

2.3.3 Inhibiting Side Channel Attacks

Public IaaS clouds use a *PA* that controls where each new VM is placed, possibly to become co-resident with other VMs sharing the same host. Common practices to secure such shared environments usually include relying on virtualisation. Virtualisation ensures strong isolation between co-resident VMs so that they become unable to interfere with each other [9]. However, virtualized isolation that completely prevents side channel attack has been proven to be difficult to achieve. The following countermeasures address side channel attacks, at the cloud provider side; the cloud user side, the hardware/software vendor side and cloud provider side respectively:

2.3.3.1 Physical Isolation Enforcement

As mentioned in Section 1.1.2, it can be argued that one pragmatic solution to mitigate side channel attacks is to disable co-residency completely. Ristenpart *et al* suggested that cloud users may consider running their VMs in physical isolation from other VMs [79]. Following this suggestion, the Amazon EC2 cloud allows users to run dedicated VMs [28], ensuring that VMs belonging to each user do not share the same physical hardware with any other cloud users' VMs.

Although this service can effectively mitigate various side channels that exist in the shared hardware, a significant price premium is required in order for cloud users to benefit from this service. It is estimated that it is 6.12 times more costly to run dedicated VMs compared to using regular VMs in Amazon EC2 [97].

One of the options left for protecting VMs from side channel attacks is only to allow other "trusted" VMs to become co-resident. If untrusted VMs become co-resident, then relocate the user's VM to another host [11]. Trusted VMs, in this case, may include VMs that are self-owned or other trustworthy VMs. This countermeasure to side channel attacks has been applied in the case of NASA and Amazon. Their

agreement over a cloud service contract gave NASA the right to run its cloud services in physically isolated, tenant-specific hardware [89]. This countermeasure requires enabling the cloud users to audit and verify the cloud provider's adherence to the policy. The researchers in [102] have introduced a promising tool called HomeAlone that uses side-channel analysis to verify that cloud providers keep their promise and disable co-residency.

However, the significant extra cost of the physical isolation of VMs diminishes its attractiveness, coupled with the fact that enabling co-residency is a definite choice of IaaS cloud providers due to its economic efficiency.

2.3.3.2 User Controlled VM Placement

The research of Ristenpart *et al* concluded that the best recommendation is to give the cloud users full responsibility and control to specify where their VMs should be placed [79]. Although this countermeasure is relatively simple, it does not solve the problem completely. In fact, it only shifts the liability to the user instead of the cloud provider without trying to eliminate the side channel or the side channel attacks.

2.3.3.3 Preventing Side Channel Vulnerabilities

Researchers have introduced a number of countermeasures that rely on side channel prevention. This can be achieved via reducing the information that can be leaked by new hardware designs or by applying various blinding techniques [36], [41].

For instance, a group of researchers at MIT [46] have recently designed a hardware chip that can hide how CPUs request information in cloud servers. This chip makes a side channel attack that uses the shared CPUs very difficult to achieve.

Apart from this hardware countermeasure, several papers discussed a number of non-hardware countermeasures that focus on preventing cache side channels. For instance, one countermeasure against cache side channel attacks that use prime and probe method (see Section 2.3.2) is to inject noise to the CPU cache timing [50], [104]. When an attacker primes the CPU cache, a special cache cleansing process is invoked. This cache cleansing process simply primes the CPU cache in order to evict the entire cache entries, and therefore preventing the attacker from gaining any useful cache timing and load measurements. However, this approach reduces the CPU cache

usefulness since it flushes the entire cache entries. Another countermeasure applies cache partitioning, where each VM is assigned a separate partition of the cache [71]. Such action ensures no cache interference among co-resident VMs. In addition, a considerable amount of literature has been devoted to introduce similar countermeasures such as adjusting each VM's perception of time, random delay insertion, using non-deterministic caches and cryptographic implementation of timing-resistant caches [31], [42], [52], [70], [72], [73], [75], [76], [91], [93]. However, [79] concluded that this type of countermeasures that rely on preventing side channel vulnerabilities suffer from two drawbacks:

First, they are typically either:

(a) Impractical, for instance incurring high overheads or nonstandard hardware is required such as [31], [41], [42], [52], [70], [71], [72], [73], [75], [91], or

(b) Application-specific [36], [50], [76], [93] or hardware-specific [46], [104].

Second, these countermeasures do not, ultimately, guarantee that all possible side channels have been anticipated and disabled, especially in the light of the increasing number of research in recent years that introduces new side channels.

Despite the efforts being paid on VMs safeguarding against existing side channels, there remains a continuous potential risk of data leakage by new side channel vulnerabilities that are yet to be discovered. Therefore, this opens an interesting research area to find an alternative approach to the reduction of the attack surface for side channel attacks, particularly one that does not rely on VMs physical-isolation or side channels prevention.

2.3.3.4 Reducing Co-residency (The Research Motivation)

This problematic coexistence of co-residency and side channel threats suggests that VMs hosted in a public IaaS cloud are exposed to side channel attacks as long as there is a non-zero co-residency probability. Thus, this particular issue motivated this thesis to look into understanding the co-residency probability in order to reduce it. The main aim in this thesis is to analyse and quantify the influence of cloud parameters (such as the number of host and users) on the co-residency probability under four commonly used *PA*s. By doing so, this thesis identifies the combination of parameters' settings in each *PA* that reduces the co-residency probability.

Unlike VMs physical-isolation and side channel prevention countermeasures, reduction of the co-residency probability does not prevent side channel. Instead, it aims to reduce the attack surface for side channel attacks by reducing the chance of co-residency (i.e. the attack avenue).

The previous sections in this chapter show that research on different aspects of co-residency has been carried out. Examples of such research include how to place VMs and detect co-resident VMs, how to exploit side channels to attack co-resident VMs and how to protect against such attacks using VMs physical-isolation and side channel prevention. However, the fundamental questions of what could effect and reduce the likelihood of co-residency occurrences in IaaS clouds and how to reduce co-residency are still not fully answered.

In a recent work, [3] highlighted the possibility of designing *PA*s that reduce the probability of co-residency. In another very recent investigation into co-residency, the focus of [10] was to formalise a new *PA* that balances between resource optimization and preventing attack VMs from co-residing with a target set of honest VMs. The proposed *PA* was shown to work best in a specific attack scenario. This scenario assumes that the attacker is computationally-bounded and that the user computation is cryptographically split among a set of VMs. Therefore, this requires the attacker to co-reside with all the user's VMs in order to steal meaningful information.

Based on the thesis aim and approach defined in Section 1.2, there are a number of important differences between the work in this thesis and the work in [10]. First, the researchers aim to minimize co-residency by formalizing a secure *PA*. Unlike their work, this thesis looks into analysing and quantifying the influence of cloud parameters on the co-residency probability in four *PA*s. Then, the appropriate settings of the most influential parameters that reduce the co-residency probability are identified in each *PA*. Therefore, their study would have been more relevant to the work in this thesis if the authors had included the cloud parameters' settings as another factor to reduce the co-residency probability. In addition, their study has assumed an attack scenario where a specific tenant's VMs are targeted by attack VMs. In contrast, this thesis extends the attack scenario to capture a worst-case scenario of a hostile or threatening cloud environment, where all VMs are targets for attackers. Moreover, the work in this thesis takes things a step further. For instance, this thesis derives two analytical estimates of the probability that the next co-residing VM belongs to an attacker.

2.4 Summary

This chapter provides a background on cloud computing and related literature. Further, the chapter considers the key issues of side channels in public IaaS clouds, and then explores common side channel attacks that can be launched against co-resident VMs. Available countermeasures against side channels, including physical isolation and side channel vulnerabilities prevention, are shown to have some drawbacks. This problematic coexistence of co-residency and side channel threats suggests that VMs hosted in a public IaaS cloud are exposed to side channel attacks as long as there is a non-zero co-residency probability. This particular issue motivated this thesis to look into understanding the co-residency probability. Therefore, this thesis quantifies the influence of cloud parameters on the co-residency probability then determines the most influential parameters. This action then helps to identify the combination of parameters' settings that reduce the co-residency probability in four commonly used *PA*s.

Unlike VMs physical-isolation and side channel prevention countermeasures, reduction of the co-residency probability does not prevent side channel attacks. Instead, it aims to reduce the attack surface for side channel attacks by reducing the chance of co-residency (i.e. the attack avenue).

Chapter 3
Models and Co-residency Behavioral Metrics

3.1 Introduction

This chapter begins by describing how the IaaS cloud is modelled in this thesis as well as the attack model. Next, the first research question is addressed by characterising the behaviour of co-residency in IaaS clouds using the co-residency metrics. The co-residency metrics are quantitative measurements that assess the characteristics of co-residency occurrence behaviour. Some of these characteristics include how likely a given honest VM *u* will be co-resided by another VM *v*, as well as how long this co-residency takes to occur. These co-residency metrics are used in Chapter 4 to identify the most influential parameters on the co-residency probability (i.e. the second research question). Moreover, these metrics are used in Chapter 5 to find the best parameter settings in each *PA* that reduce the co-residency probability (i.e. the third research question). In addition, the co-residency metrics are also used in Chapter 6 to derive analytical estimates of probabilities related to malicious co-residency.

The remainder of this chapter is organized as follows. The next section defines the system and attack models followed by Section 3.3 that outlines important notations and definitions. In Section 3.4, the co-residency metrics are defined.

3.2 System and Attack Models

This thesis considers a publicly accessible IaaS cloud wherein the *PA* allows multi-tenancy (i.e. the same physical hosts can be shared between multiple VMs) and that IaaS cloud insiders (e.g. server administrators) are trustworthy. It also assumes that confidentiality-requiring VMs of regular users (i.e. honest VMs) can receive new co-resident VMs. On the other hand, the co-residing VM *v* (either honest or attacker VM) belongs to a third-party user who can run and control a limited number of VMs simultaneously and legitimately in the cloud. This thesis considers an attacker with the following objective:

- To launch side channel attacks against arbitrary VMs. In order to do so, this requires placing a VM *v* that the attacker controls (i.e. a malicious VM) in the same host where a victim VM *u* is residing.

Before defining the system model components, the attack model can be described from two perspectives:

- **From an Honest VM's Perspective**

  Whenever a new VM *v* is placed in a given host *x*, then every honest VM *u* that has been already running and residing at *x* will experience a co-residency hit by *v*.

  Two types of co-residency hits are considered in this thesis: arbitrary co-residency hits (from malicious and honest VMs), and malicious co-residency hits (from malicious VMs only). While Chapter 4 and 5 are concerned with co-residency hits, Chapter 6's focus is on malicious co-residency hits.

- **From an Attacker's Perspective**

  This thesis considers an attack model where new honest VMs are always placed to hosts that contain either other honest VMs or no VMs at all. Therefore, the only co-residing technique (see Section 2.3.1.2) for an attacker is to place a VM *v* to become co-resident with a victim VM *u* during the latter's lifetime. Another co-residing technique can be achieved by placing *v* in a random host with the hope of *u* being placed in the same host. However, the latter co-residing technique is excluded from the scope of this thesis. Excluding this co-residing technique is based on the assumption that attackers are interested in targeting specific and existing VMs in public IaaS clouds.

  The attack model also assumes that an attacker tries to co-reside with victim VM *u* without the knowledge of where *u* is located in the cloud (i.e. brute-force placement, see Section 2.3.1.2). The latter assumption introduces a challenge to attackers, as an attacker will have to keep requesting malicious VMs with the hope that the *PA* places one of the attacker's malicious VMs to become co-resident with *u*. In addition, the *PA* that is used in the modelled IaaS cloud is assumed to be public knowledge and therefore it is known by the cloud users and the attackers. Finally, once *v* is co-resident with *u*, it is assumed that an attacker is capable of detecting that *v* has co-resided with the *u* (see Section 2.3.1.3).

The system model in this thesis consists of the following main components (see Figure 3.1):

- **Clusters:** an IaaS cloud has at least one cluster (which is a pool of hosts). The total number of clusters in the model is specified using the [Number of Clusters] parameter.
- **Hosts:** A host is a physical server that can be shared among many users to run VMs (i.e. multi-tenancy). Each host is assigned to a single cluster, where the total number of hosts in the model is specified using the [Number of Hosts] parameter. A given host is considered to be *available* when it has a free space to allocate new VMs, whereas the host becomes *full* when it has no free space to allocate new VMs. Each host can be allocated a maximum number of VMs specified using the [Max Host Utilization] parameter.
- **Users:** A user can be a normal user who runs confidentiality-requiring VMs or an attacker who aims to exploit side-channel leakage in a host (see Section 2.3.1.2) through placing VMs to become co-resident with victim VMs.
- **Users' Arrival Rate**: New users arrive in this model according to the [Users' Arrival Rate] parameter. The users' arrival rate specifies the average number of new users to arrive in every time unit. The total number of created users in an arbitrary duration of time is specified using the [Number of Users] parameter.
- **VMs:** Each user can request, run and control a limited number of VMs simultaneously. This number is specified in the model by the [Max Parallel VMs per User] parameter. Each VM terminates after a certain amount of time specified by the [VM Average Life Time] parameter. In this system model, VMs that belong to attackers will be referred to as malicious VMs.
- **VM Placement Algorithm (*PA*)**: A *PA* controls when and where (i.e. in which cluster then in which host) each new VM is placed [62]. When a VM request arrives, the *PA* selects a cluster that has at least one available host then selects an available host within that cluster to place the new VM. In case all hosts are full, no placement takes place. The system model considers four *PA*s that are used in popular IaaS cloud platforms including Eucalyptus [6], OpenNebula [60], Nimbus [85] and OpenStack [98]. These *PA*s are:
    1- First Fit.
    2- Next Fit.
    3- Power Save.

4- Random.

More detail on the above *PA*s is provided in Appendix A.

The following (Figure 3.1) shows an IaaS cloud model that has two clusters, two hosts in each cluster and three VMs placed in Host1 and Host2.

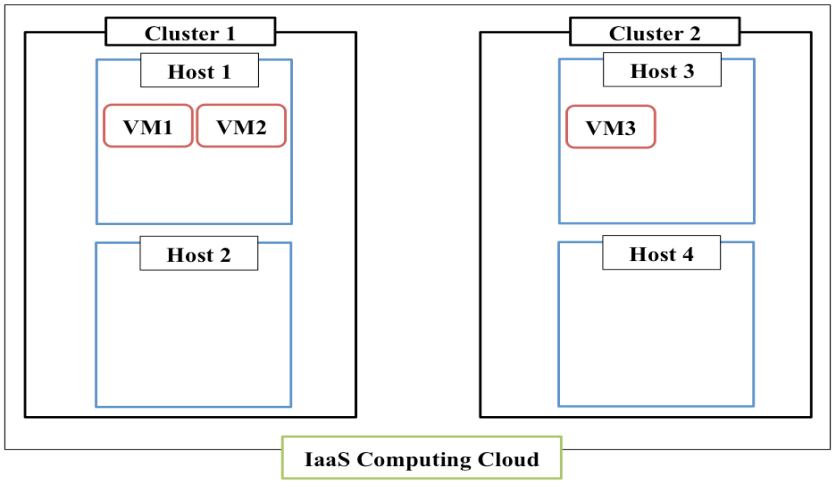


Figure 3.1 An IaaS cloud model with two clusters, two hosts in each cluster and three VMs

- **VM Request:** When a new user arrives in the IaaS cloud, the user issues a VM request to place one or more new VMs. In this system model, the [VMs per Request] parameter specifies the number of VMs in a single VM request, where this number is either [VMs per Request] or 1±[VMs per Request].. For each VM request, the *PA* places the requested VMs separately (i.e. one by one) immediately after it receives a request from the user as there is no queuing of the VM requests. As a result, the *PA* might be able to place all, part or none of the VMs for a given VM request depending on the availability of hosts. For instance, if the *PA* receives a VM request when all hosts are full, then the *PA* will not be able to place any of the requested VMs.
- **VM Requests Arrival Rate (γ):** The VM requests arrival rate, denoted as γ, represents the number of VM requests per time unit. For the convenience of the experiments, each user only issues a single VM request upon arrival. In addition, $\gamma_{malicious}$ denotes the malicious VM requests arrival rate.
- **Attackers VM Requests Ratio α**

  The attackers VM requests ratio α shows the ratio of malicious VMs to all VMs. The attackers VM requests ratio α can be defined as:

$$\textbf{attackers VM requests ratio } \alpha = \frac{\gamma_{malicious}}{\gamma}$$

- **Co-residency Conditions:** The following notations are used in the system model when describing co-residency behaviours:
    - *x* denotes a given host.
    - *v* denotes a new VM that is placed in *x* (i.e. the co-residing VM).
    - *u* denotes a given VM that resides at *x* where *v* is placed.

  There are two conditions that need to be met in order to place *v* in *x* to become co-resident with *u*:
    - *x* must have an available space to accommodate *v* when the *PA* receives the request to place *v*.
    - *v* must be requested for placement during the window of time at which the *PA* is going to select *x* for the next placement.
- **Co-residency Hit:** Whenever *v* is placed in a given host *x*, every victim VM *u* that resides at *x* will experience a co-residency hit with *v*. Co-residency hits include any hit that is received from either malicious or honest VMs during *u*'s lifetime. On the other hand, a malicious co-residency hit is a special case of the co-residency hit. A malicious hit occurs when *u* is co-resided by a malicious VM *v* that belongs to an attacker.
- **Total Number of Hits (*k*):** For a given VM *u,* the total number of hits that *u* receives is denoted as k_u, where k_u is a discrete random variable with possible values 0, 1,... . When the context is clear, the subscript *u* is dropped for convenience.
- **Total VMs in the Cloud:** During an arbitrary amount of time, the notation *n* represents the total number of placed VMs; whereas the $n_{malicious}$ is the total number of placed malicious VMs. Similarly, the notation n_{hit} is the total number of VMs that experienced at least one hit; whereas the $n_{hit\ by\ malicious}$ is the total number of VMs that experienced at least one malicious hit.
- **Time Unit:** The unit for measuring time periods in the simulation of the system model is denoted as the time unit. Examples of possible time units include second, minutes and hours. In this model, the used time unit is a minute.

3.3 Notations and Definitions

The following notations and definitions will be used throughout this thesis:

- **VM Lifetime (*LT*):** The lifetime of a VM *u* (i.e. LT_u) represents the time between the moment at which *u* is placed in a given host and the moment it is terminated.
- **Ratio (*L*) of a VM's Lifetime:** For a given VM *u* that experiences at least one hit (k>0), the entire lifetime of *u* (LT_u) can be divided into k+1 ratios L_1, L_2, ..., L_k, L_{k+1} based on the following (Figure 3.2):
  - h_0 is the time at which *u* is placed.
  - h_K is the time at which *u* experiences the K^{th} hit ($0 \leq K \leq k$).
  - h_{k+1} is the time at which *u* terminates.

With L_K as the ratio of lifetime duration between the $(K\text{-}1)^{th}$ hit and the K^{th} hit, the $L_K{}^{th}$ ratio of LT_u can be calculated as follows:

$$\boldsymbol{L_K = \frac{h_K - h_{K-1}}{LT_u}} \quad , \qquad 1 \leq K \leq (k+1)$$

The following (Figure 3.2) shows an example of the LT_u, where *u* receives *k* co-residency hits from multiple VMs $(v_1, v_2, \ldots, v_{k-1}, v_k)$ at time $(h_1, h_2, \ldots, h_{k-1}, h_k)$ respectively.

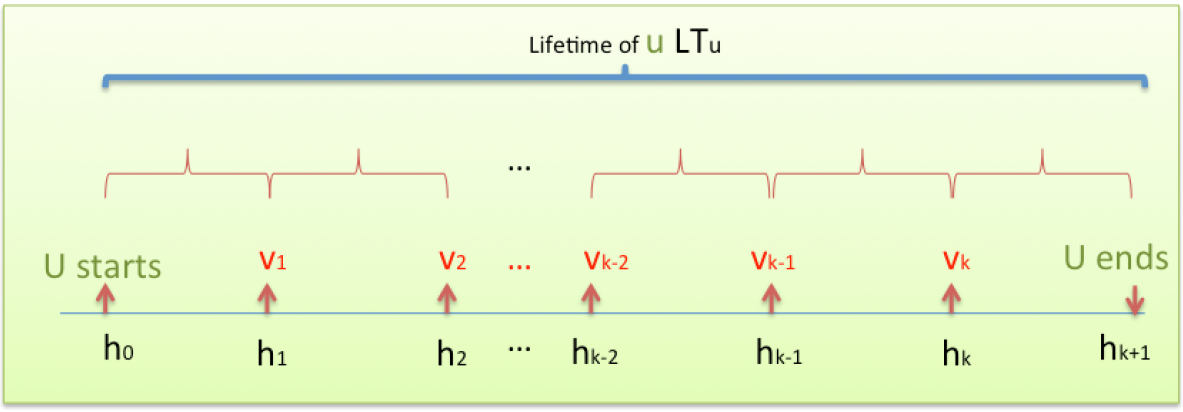


Figure 3.2 The lifetime of a VM *u* that receives *k* co-residency hits

3.4 Co-residency Metrics

Metrics defined in this section are estimated using simulation in Chapter 4 and 5. This thesis uses these co-residency metrics to:

1- Quantify the influence of multiple cloud parameters on the co-residency probability. This action helps to identify the most influential parameters in Chapter 4 (i.e. the second research question).
2- Find the best parameter settings that reduce the co-residency probability in four *PA*s in Chapter 5 (i.e. the third research question).
3- Derive analytical estimates of probabilities related to malicious co-residency in Chapter 6 (i.e. the fourth research question).

The next subsections define the co-residency metrics in detail.

3.4.1 Co-residency Coverage Probability (*CCP*)

For a given VM *u,* this metric shows the probability that *u* experiences a co-residency hit with any arbitrary VM (either malicious or honest) at least once during LT_u (i.e. P(k>0)). The Co-residency Coverage Probability *CCP* can be estimated using simulation as follows:

$$\boldsymbol{CCP = \frac{n_{hit}}{n}} \quad , \quad \boldsymbol{0 \leq CCP \leq 1}$$

For a given VM *u*, higher *CCP* indicates a higher probability of being vulnerable to at least one arbitrary co-residency hits.

3.4.2 Hit-free Lifetime Ratio (*HFL*)

For a given VM *u* that experiences at least one co-residency hit, the HFL_u represents the ratio of the time until the first co-residency hit (either a malicious or a honest hit) to the LT_u. Figure 3.3 shows how to obtain HFL_u of *u.*

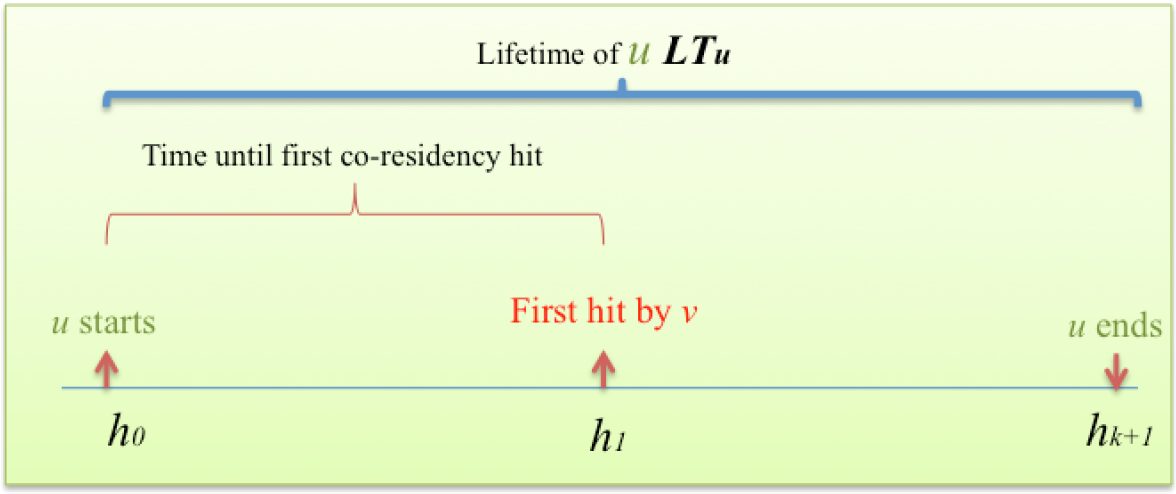


Figure 3.3 Obtaining Hit-free Lifetime Ratio HFL_u for a given VM u.

The *HFL* for a given IaaS cloud can be estimated using simulation by averaging the HFL_u of every VM u that experienced at least one hit:

$$HFL = \frac{1}{n_{hit}} \sum_{u=1}^{n_{hit}} \frac{h_{1_u} - h_{0_u}}{LT_u} \qquad , 0 < HFL \leq 1$$

3.4.3 Co-residency Vacancy (*CV*)

Figure 3.4 shows three availability windows during LT_u where host x is available to allocate new VMs. For a given VM u at x, Co-residency Vacancy CV_u is simply the ratio of the duration of these availability windows and the LT_u. In case x is full during the entire LT_u then CV_u is equal to zero ($0 \leq CV_u \leq 1$).

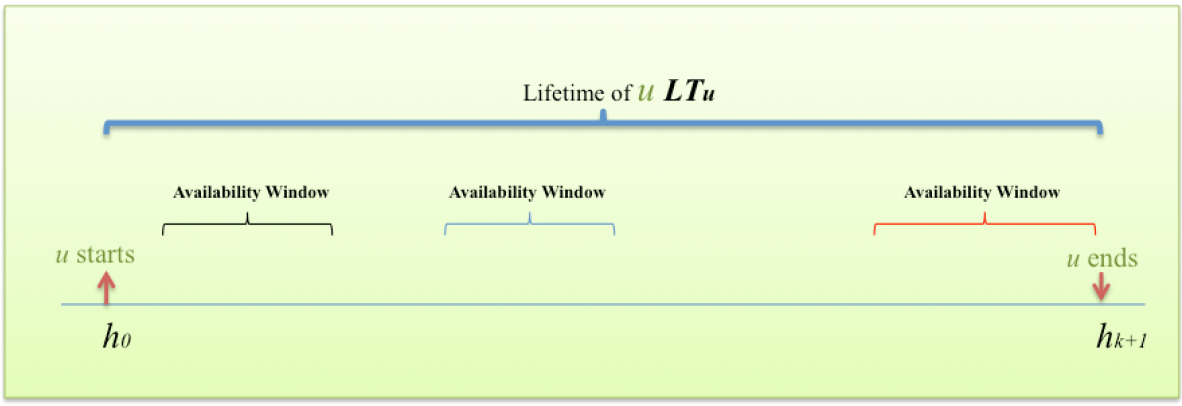


Figure 3.4 Obtaining Co-residency Vacancy CV_u of a given VM u.

The CV_u represents the ratio of the time during which VM u is vulnerable to co-residency and the LT_u. Therefore, the longer the CV_u the higher the chance that the *PA* will select u's host to place new co-residing VMs. On the other hand, it is impossible to co-reside with u when CV_u is equal to zero.

The *CV* in the cloud is estimated using simulation by averaging the CV_u of every VM u:

$$\boldsymbol{CV = \frac{1}{n}\sum_{u=1}^{n} CV_u} \qquad \boldsymbol{, 0 \leq CV \leq 1}$$

From an attacker perspective, the existence of an CV_u during LT_u (i.e. $CV_u \neq 0$) is a necessary condition to co-reside with u, however it is not sufficient to guarantee that the *PA* will select u's host to place the attacker's co-residing VMs. Therefore, $CV_u \neq 0$ represents the first condition to co-reside with u (see the conditions of a co-residency hit in Section 3.2 of this chapter), while the second condition is represented by the next metric.

3.4.4 Co-residency Activity (*CA*)

Considering the entire IaaS cloud, an inter-placement window can be defined as the time elapsed between any two consecutive placements of VMs (Figure 3.5). For a given VM u at host x, Co-residency Activity CA_u is the ratio between the inter-placement windows (that precedes each co-residency hit) and the LT_u. In case x is full during the entire LT_u then CA_u is equal to zero ($0 \leq CA_u \leq 1$).

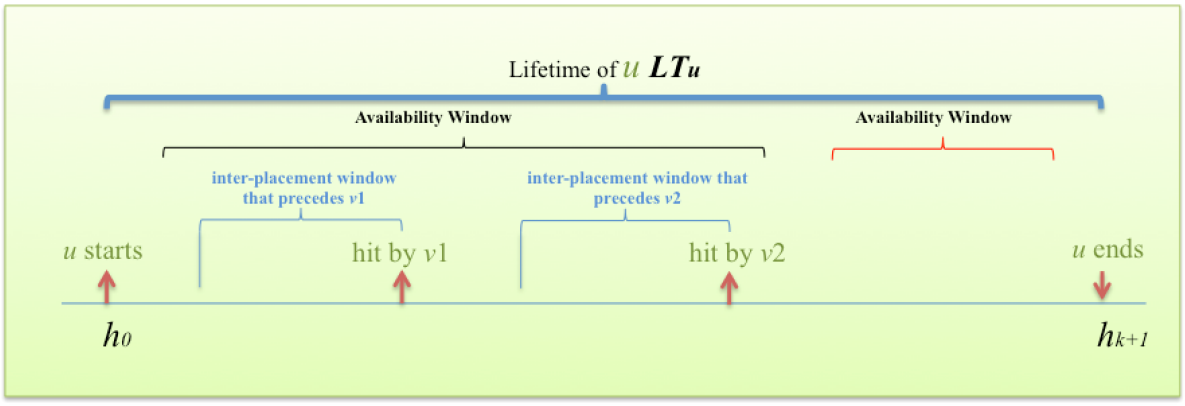


Figure 3.5 Obtaining Co-residency Activity CA_u of a given VM u.

From an attacker perspective, it is impossible to co-reside with u when the CA_u is equal to zero. Unlike the CV_u, the existence of a CA_u during LT_u (i.e. $CA_u \neq 0$) is sufficient to guarantee that u will receive a co-residency hit. Therefore, CA_u represents the second condition to co-reside with u (see Section 3.2).

Similar to the *CV* in the cloud, the *CA* is estimated using simulation by averaging the CA_u of each VM u:

$$\boldsymbol{CA = \frac{1}{n}\sum_{u=1}^{n} CA_u} \qquad \boldsymbol{, 0 \leq CA \leq 1}$$

3.5 Summary

This chapter described how the IaaS cloud is modelled as well as defining the attacker model. Then, the behaviour of co-residency in the model was characterized using four co-residency metrics. Some of these characteristics include how likely a given VM u will be co-resided by another VM v (i.e. the *CCP)*, as well as how long this co-residency takes to occur (i.e. the *HFL*). These co-residency metrics are used in Chapter 4 to identify the most influential parameters on co-residency (i.e. the second research question). In addition, Chapter 5 uses these metrics to identify the best settings of the most influential parameters in four *PA*s that reduce the probability that a given VM u experiences a co-residency (i.e. the third research question). Next, the co-residency metrics are used in Chapter 6 to derive analytical estimates of probabilities related to malicious co-residency. These probabilities include the probability that a given VM u will be co-resided by a malicious VM v and for how long it remains free from malicious hits (i.e. the fourth research question).

Chapter 4

Quantifying Influence of Cloud Parameters on Co-residency

4.1 Introduction

This chapter is dedicated to answering the second research question on the most influential cloud parameters on the co-residency metrics. A parameter's influence, measured in this chapter as the parameter's effect, is an estimate of how much varying a parameter influences the co-residency metrics (i.e. *CCP, HFL, CV* and *CA)*.

Perhaps the main challenge faced in this chapter is that there are many cloud parameters and parameters' settings to be included in limited resources experiments. In order to overcome this challenge, an Influence Evaluation Strategy is proposed to simplify the process of designing experiments that have a large number of parameters and settings. The use of fractional factorial design is one step (of multiple steps) that the strategy applies to construct a reduced and balanced experiment. Fractional factorial experiments are usually used to measure simultaneously the effects of many parameters on a product or process in a cost-effective way using minimal experimental runs [33].

Further, the Influence Evaluation Strategy proposes a statistical approach to quantify the effect of varying multiple parameters on the co-residency metrics such as varying the rate at which VMs are requested, using different numbers of hosts and others. The strategy also extends the influence evaluation to include how two parameters, together, affect the co-residency metrics (i.e. parameters interaction effect).

This chapter applies the Influence Evaluation Strategy using the *VMC* simulator as a testbed (see Appendix F for more details on the *VMC* simulator). The strategy quantifies the influence of cloud parameters on the co-residency metrics then identifies the most influential parameters and 2-parameter interactions in four *PA*s. These *PA*s are First Fit, Next Fit, Power Save and Random. The identified parameters are then used in Chapter 5 to determine the best parameters' settings that reduce the probability of co-residency in four *PA*s.

The remaining of this chapter is organized as follows. The next section outlines preliminary definitions that are used in this chapter. In Section 4.3 the four-phase Influence Evaluation Strategy is defined and then applied. Section 4.4 describes the experiments settings. The findings are presented in Section 4.5 and discussed in Section 4.6.

4.2 Preliminary Definitions

The following definitions will be used throughout this chapter:

- **Experiment Design:** In this chapter, changes are made to one or more independent variables (i.e. the parameters) in order to observe the significant effect the changes have on the co-residency metrics. Design of experiments (DoE) theory [34] provides different ways to observe these effects. In particular, the Influence Evaluation Strategy in this chapter uses fractional factorial experiment design (see Appendix B) to construct reduced and balanced experiments. Next, these experiments are used to quantify easily the influence of cloud parameters on the co-residency metrics and then to identify the most influential parameters and 2-parameter interactions in four *PA*s. The following are the main components of an experiment design: dependent variables, independent variables, levels and experimental runs.
- **Dependent Variables:** In this chapter, the Influence Evaluation Strategy uses simulation to measure the parameters and interactions effects on the co-residency metrics (*CCP, HFL, CV* and *CA*). Therefore, the co-residency metrics represent the experiments' dependent variables.
- **Independent Variables (Parameters):** The independent variables represent the experiment's input. Since the *VMC* simulator is used as a testbed, the simulator's 36 parameters (Table A.1) represent the experiments' independent variables. More detail on the *VMC* simulator is provided in Appendix A.
- **Levels**: Levels refer to the parameter's settings/values in a given experiment. The Influence Evaluation Strategy uses 2-level experiments that assign two numerical levels to each parameter: a low level and a high level.
- **Experimental runs:** In each experiment in this chapter, an experimental run consists of a unique combination of levels of parameters.
- **Fractional Factorial Experiment Design:** When there are too many parameters to be included in a limited-resources experiment, fractional factorial design helps to construct a reduced experiment design. Fractional factorial experiments are usually used to measure simultaneously the effects of many parameters on a product or process in a cost-effective way using minimal experimental runs [33]. The Influence Evaluation Strategy uses 2-way fractional factorial experiment designs that assign two levels for each parameter. As a result, this allows measuring the effects on the co-residency metrics of each parameter in isolation and in combination with another

parameter (known as 2-parameter interaction). Table 4.1 shows in (a) an example of a 2-way experiment design that has the following:

- The *CCP* metric as the dependent variable
- Two parameters X1 and X2
- 2 levels: ($X1_{low}$ and $X1_{high}$) and ($X2_{low}$ and $X2_{high}$) for X1 and X2 respectively.
- Four experimental runs.

In addition, fractional factorial design ensures a balanced experiment. A balanced experiment design guarantees that all parameters' levels are equally tested as shown in (b) and (c) in Table 4.1. More detail on how to construct 2-way fractional factorial designs is provided in Appendix B.

Run	X1	X2	*CCP*
1	$X1_{low}$	$X2_{low}$	0.55
2	$X1_{low}$	$X2_{high}$	0.53
3	$X1_{high}$	$X2_{low}$	0.34
4	$X1_{high}$	$X2_{high}$	0.39

(a) Fractional factorial experiment runs.

Run	X1	*CCP*
1	$X1_{low}$	0.55
2	$X1_{low}$	0.53

Run	X1	*CCP*
3	$X1_{high}$	0.34
4	$X1_{high}$	0.39

(b) In the left table: Runs that test X1 on Low level and on High level in the right table

Run	X2	*CCP*
1	$X2_{low}$	0.55
3	$X2_{low}$	0.34

Run	X2	*CCP*
2	$X2_{high}$	0.53
4	$X2_{high}$	0.39

(c) In the left table: Runs that test X2 on Low level and on High level in the right table

Table 4.1 Testing each parameter's level in a two-way fractional factorial experiment design

- **Range:** A range measures the numerical distance between a parameter's 2-level values (i.e. the low and high values). Two types of ranges are defined for each parameter: the *narrow* range and the *broad* range, where the narrow range is nested

within the broad range. Table 4.2 shows how the [Number of Clusters] parameter, for instance, is tested using narrow range and broad range.

Range	Narrow Range		Broad Range	
	Low Level	High level	Low Level	High level
Number of Clusters	15	30	10	50

Table 4.2 Example of the narrow and broad ranges.

- **Narrow-experiment and Broad-experiment:** The Influence Evaluation Strategy uses two fractional factorial experiments (i.e. narrow-experiment and broad-experiment) that will be conducted using four *PA*s. The narrow-experiment refers to the experiment that assigns two levels to the parameters from the narrow range, whereas the broad-experiment assigns two levels from the broad range. Each experiment consists of 16 experimental runs that are conducted using simulation.
- **Parameter Effect:** A parameter's effect is an assessment of the parameter influence. The parameter's effect measures the size of the change on the co-residency metric that occurs when the parameter level is varied. In the Influence Evaluation Strategy, ANOVA test calculates the effects using the simulation estimates of the co-residency metrics (see Section 4.3.4.1).
- **2-Parameter Interaction's Effect:** In addition to the parameters' effects, the effect of every 2-parameter interaction is evaluated. Two parameters interact if the effect of one of the parameters differs depending on the level of the other parameter. For instance, the effect of users' arrival rate on the co-residency metrics could differ depending on how many VMs each user requests. An interaction's effect measures the size of the change on the co-residency metric that occurs when the levels of two parameters (combined) are varied. An interaction of two parameters X1 and X2 is denoted as X1*X2.
- **Effect's Level of Significance:** The significance level of an effect can be reported in the following three ways based on the p-value as suggested by [22]:
  - `significant': $0.01 < \text{p-value} < 0.05$;
  - `highly significant': $0.001 < \text{p-value} < 0.01$; and
  - `very highly significant': $\text{p-value} < 0.001$

This Influence Evaluation Strategy reports an effect to be significant if the effect has a p–value that is less than 0.05. Only statistically significant effects are considered when quantifying the parameters influence (see Section 4.3.4.2).

- **Experimental Runs Repetitions:** A repetition of an experimental run is used to increase the confidence on the results and to reduce the possibility of errors or anomalous results [14]. In this chapter, each experimental run is simulated in ten repetitions.
- **Design Resolution:** The resolution of a fractional factorial experiment design specifies the degree to which the effect of each parameter confounds with the effects of other parameters and interactions (see Appendix B for more details). A fractional factorial design's resolution can be of any of the following types:
  - II: A parameter's effect is confounded with another parameter's effect.
  - III: A parameter's effect may confound with a 2-parameter interaction's effect.
  - IV: A parameter's effect does not confound with any other parameter's effect nor with any 2-parameter interactions' effect.
  - V: A parameter's effect does not confound with any 3-parameter interactions' effect, and a 2-parameter interactions' effect does not confound with any 2-parameter interactions' effect.
  - VI: A parameter's effect does not confound with any 4-parameter interactions' effect, and a 2-parameter interactions' effect does not confound with any 3-parameter interactions' effect.

A resolution IV fractional factorial design is used throughout this chapter.

4.3 Influence Evaluation Strategy

Using the *VMC* simulator, the Influence Evaluation Strategy quantifies the influence of cloud parameters on the co-residency metrics in four *PA*s (First Fit, Next Fit, Power Save and Random). Then, the strategy identifies the most influential parameters and 2-parameter interactions in the four *PA*s. The *VMC* simulator requires 36 different parameters to be defined in order to simulate the IaaS cloud model in Chapter 3. Examining each parameter effect under many levels could increase the reliability of the results, as well as increasing the experiment size. However, measuring the effects of all the 36 parameters using every possible level is impractical due to the limited time and resources available to this thesis. The

Influence Evaluation Strategy overcomes this challenge by simplifying the process of conducting experiments that have a large number of parameters and levels.

This simplification tries to obtain a reduced-size experiment by reducing the number of parameters, levels, and experiment's runs as much as possible without seriously affecting the experiment's outcome. More precisely, the strategy comprises an effective reduction of parameters in the first phase and a parameters levels reduction in the second phase leading to the design of two reduced-size experiments using fractional factorial design in the third phase. Finally, the fourth phase uses the *VMC* simulator to conduct the experiments. Ultimately, the Analysis Of Variance test ANOVA is applied to quantify the parameters' effects on the co-residency metrics both in isolation (i.e. parameters' effects) and combination (i.e. 2-parameter interactions effects).

The first three phases of the strategy are extended from [59] with slight differences that are indicated throughout this chapter.

This Influence Evaluation Strategy consists of four phases that aim to:

1- **Simplify** the process of designing experiments that have a large number of parameters and levels. As shown in the following sections, this simplification tends to reduce the number of parameters and levels and, therefore, the experiment size as much as possible without affecting the experiment's outcome.

2- **Identify** the parameters and interactions that influence the co-residency metrics the most in IaaS clouds under each *PA*. In order to do so, the strategy quantifies the influence of multiple cloud parameters and interactions on the co-residency metrics.

The most influential parameters and interactions on the co-residency metrics will be used in Chapter 5 to identify the best parameters' settings in four *PA*s that reduce the co-residency probability.

The following sections outline the four phases of the Influence Evaluation Strategy and how they have been applied to design and conduct two reduced size experiments: the narrow-experiment and the broad-experiment.

4.3.1 Phase 1: Parameters Reduction Using Composed Parameters

Phase input: 36 parameters used by the *VMC* simulator.

Phase Output: eight composed parameters.

The parameters reduction using the parameter composing method was originally presented in [59] as one of the multiple steps towards designing reduced size experiments for identifying the most significant parameters influencing large-scale model behaviour. Using the parameters of the Koala simulator, they demonstrated the efficiency of composing similar parameters and reduced the Koala's parameters from 82 to only 23.

This parameters reduction method was applied to the *VMC* simulator's input parameters. The parameters reduction method composes parameters that describe similar characteristics to form a single parameter, referred to as a composed parameter. Table 4.3 shows the *VMC* simulator's parameters after the parameters' reduction.

For example, the Number of Hosts parameter X2 in Table 4.3 composes five similar parameters. That is Number of Hosts of Type H1, Number of Hosts of Type H2, Number of Hosts of Type H3, Number of Hosts of Type H4 and Number of Hosts of Type H5 respectively. Each of the previous individual parameters specifies the number of hosts for a single host type, whereas the composed parameter X2 specifies the total number hosts of all types combined. Another example is the Maximum Host Utilization parameter X3. Again, X3 composes similar parameters that individually specify the maximum utilization limit for each host type in the *VMC* simulator. The parameters reduction phase was applied to the *VMC* simulator parameters and successfully reduced the number of parameters that will be used in the experiments from 36 to 8 parameters. Out of these eight parameters, Number of Clusters X1 is the only non-composed parameter as there is no similar parameter to be composed with.

On the other hand, [59] continued the parameters reduction and reduced another 12 parameters of the Koala simulator using domain knowledge to eliminate the parameters that appear to be insignificant to the intended experiment. While this step seems to further reduce the number of parameters, it is not applied in this thesis. This is because this step requires prior knowledge of the parameters that influence the co-residency metrics, and such knowledge is what this thesis is trying to discover.

Composed Parameters	ID	Description	Composed Parameter Consists of these Parameters
Number of Clusters	X1	How many clusters to be created in the simulated model. A cloud has at least one cluster that contains a pool of hosts.	N/A
Number of Hosts	X2	A cluster has at least one host. A host is a physical server that runs VMs. Each host is assigned to a single cluster, where the total number of hosts in the IaaS cloud is specified using the [Number of Hosts] parameter. Each host can be allocated a limited number of VMs specified using the [Max Host Utilization] parameter. Hosts will be distributed randomly into clusters with equal probability.	Number of Hosts of Type ***H1***
			Number of Hosts of Type ***H2***
			Number of Hosts of Type ***H3***
			Number of Hosts of Type ***H4***
			Number of Hosts of Type ***H5***
Max Host Utilization	X3	A Host is Full when the hosted VMs usage of the host's resources (CPU, memory and storage) reaches the Max Host Utilization percentage.	Max Utilization for Host Type ***H1***
			Max Utilization for Host Type ***H2***
			Max Utilization for Host Type ***H3***
			Max Utilization for Host Type ***H4***
			Max Utilization for Host Type ***H5***
Users' Arrival Rate	X4	New users arrive into the IaaS cloud according to the [Users' Arrival Rate] parameter that represents the average number of new users to be created every time unit.	Users' Arrival Rate Of Type ***U1***
			Users' Arrival Rate Of Type ***U2***
			Users' Arrival Rate Of Type ***U3***
			Users' Arrival Rate Of Type ***U4***
			Users' Arrival Rate Of Type ***U5***
Max Number of Users	X5	The maximum number of users (of all types) to be created during the simulation	Max Number of Users of Type ***U1***
			Max Number of Users of Type ***U2***
			Max Number of Users of Type ***U3***
			Max Number of Users of Type ***U4***
			Max Number of Users of Type ***U5***
Max Parallel VMs per User	X6	The maximum number of concurrently running VMs (of all types) a single user can have	Max Parallel VMs of User Type ***U1***
			Max Parallel VMs of User Type ***U2***
			Max Parallel VMs of User Type ***U3***
			Max Parallel VMs of User Type ***U4***
			Max Parallel VMs of User Type ***U5***
VM Average Life Time	X7	How long a user (on average) holds his running VM (of any type) before terminating it (in time units)	***X_SMALL*** VM Average Lifetime
			SMALL VM Average Lifetime
			MEDIUM VM Average Lifetime
			LARGE VM Average Lifetime
			X_LARGE VM Average Lifetime
VMs per Request	X8	The number of VMs to be created in each new VMs request The number of VMs per request must be less than or equal to X6.	VMs per Request Rate for User Type ***U1***
			VMs per Request Rate for User Type ***U2***
			VMs per Request Rate for User Type ***U3***
			VMs per Request Rate for User Type ***U4***
			VMs per Request Rate for User Type ***U5***

Table 4.3 The *VMC* parameters after reduction

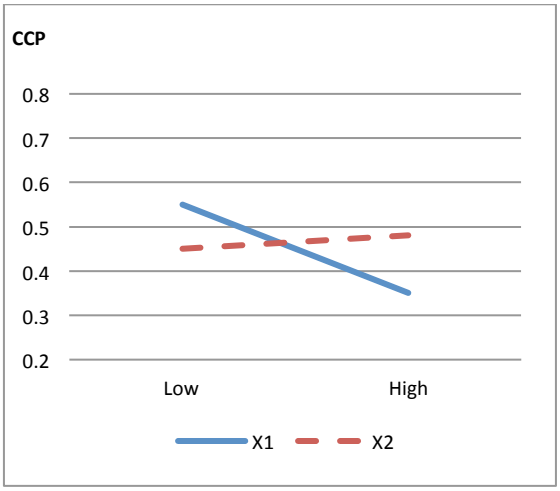
4.3.2 Phase 2: Levels Reduction Using Ranges

Phase inputs: eight parameters from the first phase.

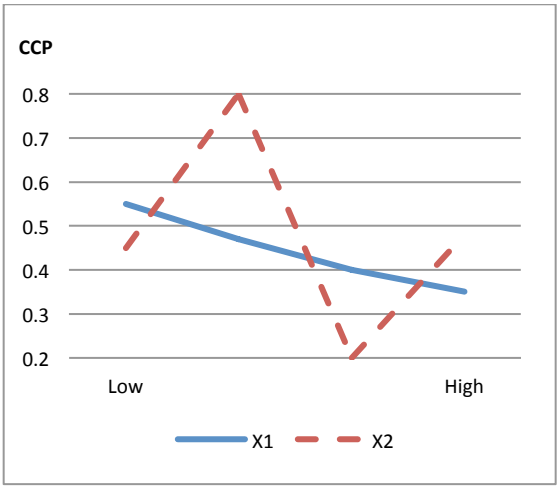
Phase Outputs: Per parameter: two levels for the narrow range and two levels for the broad range.

Having reduced the number of parameters to eight in the first phase, this phase of the Influence Evaluation Strategy reduces the number of the parameters' levels. To highlight the challenge faced in this phase, an assumption can be made that each parameter can take up to 2^{32} levels (i.e. the maximum value for a 32-bit unsigned integer). Moreover, each parameter needs to be tested in each possible level to measure its effect. Consequently, this would result in a gigantic experiment design that consists of $(2^{32})^8$ experimental runs, which exceeds the time and resources available for this thesis. Again, [59] suggested a solution to this challenge, one that assigns two levels for each parameter (low and high values). Then simply measures the effect between these two levels and verifies the effect's statistical significance. There are a number of advantages of using two levels to measure the effect. Firstly, it requires less experimental runs to test all parameters combinations, which allows more repetitions of the experimental runs that can contribute significantly to the experiment's robustness. In addition, using two levels per parameter fits naturally towards the next phase of this Influence Evaluation Strategy that uses 2-way fractional factorial designs to reduce the experiment size in a structured and balanced fashion.

However, using two levels to measure the effects comes with its limitations. Firstly, the two levels that will be used to test each parameter's effect, obviously, do not cover every possible level. Secondly, measuring the effect of a parameter that is varied between two levels does not guarantee that the parameter has a linear effect on the tested two levels. To illustrate the effect's linearity issue, Figure 4.1 represents the data from Table 4.1. Figure (a) suggests that varying the two levels of parameter X2 does not change the *CCP* metric as much as X1 does. However, testing X1 and X2 at more levels between the original low and high levels can reveal a contradicting result as (b) demonstrates that X2 has, in fact, a larger influence, changing the *CCP* more than X1 does.



(a) X1 and X2 tested at two levels: low and high, suggesting that X2 does not change the *CCP* metric between its two levels as much as X1 does.



(b) X1 and X2 tested at extra two levels between low and high, revealing that X2 has more influence as it changes the *CCP* more than X1.

Figure 4.1 Limitation of using two levels to test the parameters' effects.

The previous two limitations are addressed in the Influence Evaluation Strategy. Specifically, the strategy tests the effect of each parameter on the co-residency metrics twice using two ranges: (1) the narrow range and (2) the broad range (see range definition in Section 4.2). Each parameter will be tested at two levels per range such that the narrow range is a subset of the broad range. Such action helps the exploration of a wider range of parameters levels. Another workaround to address the second limitation is carried out in Chapter 5. Firstly, the most influential parameters on the co-residency metrics are tested at ten levels. Secondly, Pearson's correlation analysis [14] is used to ensure that the most influential parameters have significant linear effects with the co-residency metrics. It is worthwhile to mention that this linearity check is not present in the Mills method. This linearity check will also evaluate the accuracy of the proposed Influence Evaluation Strategy (see Section 5.3.5 for more details). For each parameter, Table 4.4 outlines the selected two levels for each range. Where possible, the broad range extends the distance between each parameter's two levels compared to the narrow range. For instance, the Number of hosts X2 in the broad range examines two extremes in terms of cloud infrastructure size (100 hosts in low level to 30000 hosts in high level) compared to the narrow range (1000 hosts to 10000 hosts). Similarly, the same is applicable to the rest of the parameters.

It is important to note that public IaaS cloud providers, such as Amazon EC2 and Windows Azure, usually obscure the details of their cloud infrastructure, networks and even *PA*s [79]. For this reason and based on the available literature [10], [58], [59], the two ranges for each parameter were selected in such a way that they represent different variations of possible IaaS settings in the real world.

Parameter	**ID**	**Narrow Range**		**Broad Range**	
		Low level	**High level**	**Low level**	**High level**
Number of Clusters	X1	15	30	10	50
Number of Hosts	X2	1000	10000	100	30000
Max Host Utilization	X3	80%	90%	50%	100%
Users' Arrival Rate	X4	2	3	1	5
Number of Users	X5	35000	50000	10000	75000
Parallel VMs per User	X6	12	18	5	20
VM Average Lifetime	X7	2000	2500	1600	3600
VMs per Request	X8	2	3	1	4

Table 4.4 The selected two levels per range for each parameter

4.3.3 Phase 3: Experiment Reduction Using Fractional Factorial Design

Phase inputs: from the first phase : eight composed parameters, and
from the second phase: two levels for the narrow range per parameter
and
two levels for the broad range per parameter

Phase Output: two fractional factorial experiments: narrow-experiment and
broad-experiment

This phase of the Influence Evaluation Strategy uses the eight composed parameters and their two levels ranges to design the experiments that will be used to quantify the influence of cloud parameters on the co-residency metrics. Each experiment is designed so that it tests all necessary parameter combinations, including parameter interactions while trying to reduce the number of experimental runs. However, testing every parameter combination makes the number of experimental runs grow very quickly. 2-way fractional factorial design (defined in

detail in Appendix B) will be used to construct balanced experiments with minimum experimental runs in order to overcome this challenge. A balanced experiment design ensures that all parameter levels are equally tested, like the example shown in (b) and (c) in Table 4.1. The basic concept of fractional factorial design is to include a subset (fraction) of the original experimental runs such that only the important parameter combinations and interactions are covered. This is in contrast to the traditional one parameter at a time experimental approaches [90]. A resolution IV fractional factorial design is used throughout this chapter. A resolution IV design ensures that the effect of a given parameter does not confound with other parameters and 2-parameter interactions effects. The following sections describe how fractional factorial is used to design the experiments in this chapter.

4.3.3.1 Factorial Experimental Designs

One of the main objectives of experimental design is to construct an experiment that is capable of generating accurate results to support or reject the research hypothesis [59]. A good experimental design must include all the necessary parameters combinations in order to allow balanced experimentations. However, adding more parameters makes the experiment's design grow very quickly. For instance, the experiment in this chapter includes eight parameters where each parameter takes two possible levels per range, yielding a total of (2^8 = 256) experimental runs (i.e. parameter combinations). In Design of Experiment (DoE) theory, this type of experimental design where all parameter combinations are tested is known as full factorial design. However, one of the challenges of using full factorial design is that it can be difficult to test every possible parameter combination and to repeat the experiment at the same time. For instance, including eight parameters with two levels in a full factorial experiment with ten repetitions dramatically increases the number of the experimental runs to 2560. One practical solution to overcome this limitation is to apply fractional factorial to design reduced size experiments. Appendix B provides a full description of factorial and fractional factorial experimental design. Section B.2 shows the steps to design the main fractional factorial experiment in this chapter that uses a $\frac{1}{2^4}$ fraction of the 2^8 full factorial experiment, reducing the experimental runs from 256 to 16 runs only. This design, of resolution IV, is denoted as 2_{IV}^{4}.

Using the fractional factorial design in Table B.4, levels from the narrow and broad ranges (Table 4.4) are assigned to the fractional factorial design table. This results in the narrow-

experiment that uses the 2-level values of the narrow range (Table 4.5) and the broad-experiment that uses the 2-level values of the broad range (Table 4.6).

Run	X1	X2	X3	X4	X5	X6	X7	X8
1	15.0	1000	80	2.0	35000	12	2000	2.0
2	30.0	1000	80	2.0	35000	18	2500	3.0
3	15.0	10000	80	2.0	50000	12	2500	3.0
4	30.0	10000	80	2.0	50000	18	2000	2.0
5	15.0	1000	90	2.0	50000	18	2500	2.0
6	30.0	1000	90	2.0	50000	12	2000	3.0
7	15.0	10000	90	2.0	35000	18	2000	3.0
8	30.0	10000	90	2.0	35000	12	2500	2.0
9	15.0	1000	80	3.0	50000	18	2000	3.0
10	30.0	1000	80	3.0	50000	12	2500	2.0
11	15.0	10000	80	3.0	35000	18	2500	2.0
12	30.0	10000	80	3.0	35000	12	2000	3.0
13	15.0	1000	90	3.0	35000	12	2500	3.0
14	30.0	1000	90	3.0	35000	18	2000	2.0
15	15.0	10000	90	3.0	50000	12	2000	2.0
16	30.0	10000	90	3.0	50000	18	2500	3.0

Table 4.5 The narrow-experiment design

Run	X1	X2	X3	X4	X5	X6	X7	X8
1	10	100	50	1	10000	5.0	1600	1.0
2	50	100	50	1	10000	20.0	3600	4.0
3	10	30000	50	1	75000	5.0	3600	4.0
4	50	30000	50	1	75000	20.0	1600	1.0
5	10	100	100	1	75000	20.0	3600	1.0
6	50	100	100	1	75000	5.0	1600	4.0
7	10	30000	100	1	10000	20.0	1600	4.0
8	50	30000	100	1	10000	5.0	3600	1.0
9	10	100	50	5	75000	20.0	1600	4.0
10	50	100	50	5	75000	5.0	3600	1.0
11	10	30000	50	5	10000	20.0	3600	1.0
12	50	30000	50	5	10000	5.0	1600	4.0
13	10	100	100	5	10000	5.0	3600	4.0
14	50	100	100	5	10000	20.0	1600	1.0
15	10	30000	100	5	75000	5.0	1600	1.0
16	50	30000	100	5	75000	20.0	3600	4.0

Table 4.6 The broad-experiment design

As pointed out at the beginning of this chapter, the outcome of the Influence Evaluation Strategy is to quantify the influence of cloud parameters on the co-residency metrics in four *PA*s. Then, the strategy will be able to identify the most influential parameters and 2-parameter interactions. Therefore, the next phase uses the *VMC* simulator to estimate the co-residency metrics using the previous narrow-experiment (Table 4.5) and the broad-experiment (Table 4.6) under the four *PA*s. In addition, each experimental run is simulated ten times to increase the confidence in the findings. This results in 320 simulations per *PA* and 1280 simulations in total for the four *PA*s.

4.3.4 Phase 4: Quantifying the Parameters Influence on the Co-residency Metrics

Phase inputs: From phase 3: two fractional factorial experiments:
narrow-experiment and
broad-experiment

Phase Output: Most influential parameters and interactions on the co-residency metrics under each *PA*

Using the *VMC* simulator, this phase quantifies the influence of cloud parameters on the co-residency metrics then identifies the most influential parameters and 2-parameter interactions in four *PA*s. This action aims to answer the research's second question on what influences co-residency the most. Parameters and interactions influence is quantified under each *PA* separately in order to make the identification process more accurate since the *PA* is responsible for controlling where and when each VM is placed in the cloud. This separation is essential in this thesis to examine whether an influential parameter under a given *PA* would have the same, less, more or no influence at all under another *PA*. Therefore, the narrow-experiment and broad-experiment will be simulated in this phase using four *PA*s (i.e. First Fit, Next Fit, Power Save and Random). Before describing how the simulation experiments in this phase were carried out, the following sections explain how to measure and quantify the effect and how to determine the effect significance.

4.3.4.1 Effect Definition

As defined at the beginning of this chapter, changing a parameter's level can yield a change on a given co-residency metric, where the size of this change represents the parameter's effect on that metric. The effect of a parameter or an interaction X on a given co-residency metric *M* is calculated as follows:

Estimated Effect of X = $| \overline{\boldsymbol{M}}_{\boldsymbol{High}} - \overline{\boldsymbol{M}}_{\boldsymbol{Low}} |$

where:

$\overline{M}_{High}$ = Metric's average when X is on its high level

$\overline{M}_{Low}$ = Metric's average when X is on its low level

An effect also tells the direction of the change. For instance, a negative effect implies that changing the parameter's level from high to low yields a decrement in the co-residency metric value. However, this chapter focuses on quantifying the parameters and interactions

effects, regardless of the direction of the effects. Therefore, the magnitude of the effect value is used to quantify the parameters and interactions influence on the co-residency metrics. A higher effect value implies that a larger change takes place on the co-residency metric and vice versa.

The effect for each parameter and interaction can be easily measured using Analysis Of Variance (ANOVA). ANOVA is a collection of statistical models that can be used to measure the effect of a single parameter as well as the effect of a 2-parameter interaction. Using Minitab statistical software [7], an ANOVA test was applied on the simulation estimates of the co-residency metrics. These estimates were obtained from the narrow-experiment and the broad-experiment for each of the four *PA*s (First Fit, Next Fit, Power Save and Random). The ANOVA test shows the effect of each parameter and interaction relating to each co-residency metric. Using Minitab, the following Table 4.7 displays a example of ANOVA test output of the effects on the *CCP* metric for two parameters X1 and X2 and their 2-parameter interaction.

```
------------------------------------------
Fractional Factorial Design

Estimated Effects and for CCP (coded units)
------------------------------------------
Term       Effect         P
X1        -0.035          0.000
X2        -0.0131         0.120
X1*X2      0.0751         0.890
------------------------------------------
```

Table 4.7 Minitab statistical software output example

Having defined how to calculate the effects, it is more important to verify that the calculated effect is significant enough to reproduce the same change on the co-residency metrics. The effect significance can be verified using the p-value of each effect. The following section defines when to consider the effect to be significant.

4.3.4.2 Effect Significance

In order to accept that a parameter or an interaction has a significant affect on a given metric, a null hypothesis H_0 and an alternative hypothesis H_1 are defined as follows:

$$\boldsymbol{H_0}\textbf{: effect} = \mathbf{0}$$

$$\boldsymbol{H_1}\textbf{: effect} \neq \mathbf{0}$$

The above null hypothesis H_0 implies that there is no effect. In order to calculate the p-value for H_0 testing, Student's t-test is used [14]. The effect's p-value gives the probability that H_0 is held true when the experiment is conducted again. In this thesis, an effect is considered to be statistically significant when there is less than 5% chance of accepting H_0 whenever the experiment is repeated. Therefore, if the effect's p-value is below 0.05 then H_0 is rejected (and therefore H_1 is accepted), and the effect is considered to have a statistical significance. In the previous example in Table 4.7, parameter X1 has a significant effect since its p-value is less than 0.05. On the other hand, X2 and the X1*X2 interaction do not have significant effects as their p-values are greater than 0.05.

In order to quantify the parameters' influence under a given *PA*, the statistically significant effects on each metric are used to calculate the overall Weighted Effect. The overall Weighted Effect quantifies the overall influence of each parameter and interaction on the four co-residency metrics combined as shown in the next section.

4.3.4.3 Overall Weighted Effect *WE*

For each *PA*, the previous step defines how to measure the parameters and interactions effects and more importantly how to verify the effects significance. The parameter/interaction overall Weighted Effect, or overall *WE* for short, consists of the parameter/interaction *WEs* on the four co-residency metrics. A parameter's *WE* on a given co-residency metric quantifies the parameter's effect relative to other parameters/interactions effects on the same metric. More precisely, the *WE* measures an effect with respect to the maximum observed significant effect on the same metric. As each *PA* is tested using a narrow-experiment and a broad-experiment, the parameter/interaction *WE* on a given co-residency metric M is calculated from both experiments as follows:

For a given parameter or interaction X, let *e* be a variable that takes the following values (where the p-value corresponds to the parameter or interaction effect):

$$\boldsymbol{e} = \begin{cases} \dfrac{\text{X's effect on M}}{\text{Maximum significant effect of all Xs on M}}, & \text{p} - \text{value} < 0.05 \\ 0, & \text{p} - \text{value} \geq 0.05 \end{cases}$$

Then X's *WE* on a given metric M =

e in the narrow-experiment

+

e in the broad-experiment, $0 \leq WE \leq 2$

For each X under a given *PA*, the sum of *WE*s on all four co-residency metrics represents X's overall *WE* on the co-residency metrics:

X's Overall *WE* = ∑(X's *WE* on each M), $0 \leq$ Overall *WE* ≤ 8

The maximum *WE* a parameter/interaction can achieve on a given metric is two. This is only possible when the parameter/interaction has the highest significant effect on that metric in the narrow-experiment and the broad-experiment together. Accordingly, the maximum overall *WE* a parameter/interaction can achieve under a given *PA* is eight, given that the parameter/interaction achieves maximum *WE* (i.e. two) on the four co-residency metrics. Therefore, the overall *WE* for each parameter/interaction quantifies its overall influence on the co-residency metrics compared to other parameters and interactions.

Under each *PA*, the parameters and interactions with the highest overall *WE* are selected as the most influential parameters. The selected parameters will be used in Chapter 5 to answer the third research question on the best parameters' settings that reduce the co-residency probability in four *PA*s.

4.4 Experimental Setup

The *VMC* simulator has been used to estimate the co-residency metrics in the narrow-experiment and the broad-experiment (Table 4.5 and Table 4.6) under four *PA*s: First Fit, Next Fit, Power Save and Random. Each experiment consisted of 16 experimental runs that tested each parameter eight times per level. In addition, each experimental run was repeated in ten simulation repetitions per *PA*. Such an approach allowed obtaining 160 test observations per parameter level per *PA*, and, therefore, increased the confidence in the simulation results.

All experimental runs were simulated for a period of 3800 minutes, and the simulation results were collected after a 200 minutes warm-up period. This warm-up period allowed an opportunity for VM placement activities to take place before recording the results. Moreover, the *VMC* simulator depends on Java's random function to simulate the co-residency behaviour. This function accepts a number and returns a pseudorandom, uniformly distributed value between 0 (inclusive) and the specified number (exclusive). The sequence of the returned random numbers depends on the function's seed number. The seed number sets the initial value of the internal state of the pseudorandom number generator. If two simulation runs are using the same random seed, they will generate identical sequences of numbers. In order to enhance the robustness of the random numbers generated using Java's random function, the system clock is used as a seed number each time the random function is used. Such an approach increased the confidence that each simulation run will receive a different sequence of the generated random numbers. In addition, the simulation runs were conducted at different times of the day using multiple PCs that have different configurations (Table 4.8). Such an action ensures that the time at which the simulation is carried out, and the PCs configurations' impact on the simulation results, is minimal.

Configurations	**Type 1**	**Type 2**	**Type 3**
Processor Spec.	Intel(R) Core i7 CPU	Intel(R) Core i7-3770 CPU	Intel(R) Core i5 CPU
No. of Cores	8 CPUs X 2.93GHz	8 CPUs X 3.4GHz	2 CPUs X 2.4GHz
Memory Size	4096MB RAM	8192MB RAM	8192MB RAM
Operating System	Windows 7 (64 bit)	Windows 7 (64 bit)	OS X 10.9.4

Table 4.8 PC configurations used to run the *VMC* simulator

The *VMC* simulator generates the results in Microsoft Excel format and text format (see Appendix A). Once the simulation is done, the results are entered into the Minitab statistical software to carry the ANOVA test in order to measure the parameters and interactions effects.

4.5 Findings

The influence of cloud parameters and interactions on the co-residency metrics were quantified under each *PA* as follows:

1 For each parameter and interaction: the effect and its significance on each of the co-residency metrics (*CCP, HFL, CV* and *CA*) were calculated.
2 Then, the overall *WE* of each parameter and interaction was calculated to quantify the parameters and interactions influence on the co-residency metrics.

Under each *PA*, four parameters and interactions with the highest overall *WE* were identified as the most influential parameters on the co-residency metrics.

The next two subsections present the results in an orderly sequence using the above two steps. The presented results confirm the thesis's first hypothesis by quantifying the parameters' and interactions' influence on the co-residency metrics. The following observations were made as a result of simulating all experimental runs from the narrow-experiment and the broad-experiment (Table 4.5 and Table 4.6). Each run was tested in ten simulation repetitions under four *PA*s: First Fit, Next Fit, Power Save and Random.

4.5.1 Significant Effects Results

For the narrow-experiment and the broad-experiment, Table 4.9 shows the p-values (i.e. the level of significance) of the parameters and 2-parameter interactions effects on the four co-residency metrics under each *PA*.

Under each *PA*, each level of every parameter was tested in 80 simulation repetitions in both the narrow-experiment and the broad-experiment, and, therefore, each effect's p-value was calculated with 79 degrees of freedom. Wherever a parameter's effect is significant (i.e. has a p-value < 0.05) then the effect will be considered in the parameter's overall *WE* calculations.

		Effects' p-values on the co-residency metrics per Placement Algorithm							
PA↓	**Metrics →**	***CCP***		***HFL***					
	Experiment → **Parameters ↓**	**Broad**	**Narrow**	**Broad**	**Narrow**	**Broad**	**Narrow**	**Broad**	**Narrow**
First Fit	**X1**	0	0.897	0.014	0.681	0	0.532	0	0.698
	X2	0	0.108	0.01	0	0	0	0	0
	X3	0	0	0.074	0.549	0.035	0	0	0.87
	X4	0	0	0	0.001	0.511	0.278	0	0
	X5	0	0.565	0	0.811	0	0.442	0.004	0.643
	X6	0.113	0.789	0.029	0.686	0.482	0.485	0.017	0.021
	X7	0	0	0	0.226	0	0.409	0	0
	X8	0	0	0.251	0.226	0.002	0.984	0	0.16
	X1*X2	0	0.187	0.046	0.687	0.209	0.704	0	0.823
	X1*X3	0	0.036	0.027	0.119	0	0.934	0	0
	X1*X4	0.16	0.863	0.05	0.88	0	0.406	0	0.012
	X1*X5	0.126	0.508	0.061	0.619	0.004	0.588	0.131	0.004
	X1*X6	0.111	0.978	0.826	0.743	0.04	0.32	0.001	0.716
	X1*X7	0	0.594	0.006	0.663	0	0	0	0.773
	X1*X8	0	0.699	0.271	0.469	0.027	0.528	0	0
Next Fit	**X1**	0	0.147	0	0.824	0.632	0.9	0	0.323
	X2	0	0	0	0	0	0	0	0
	X3	0	0.252	0.255	0.831	0.352	0.323	0	0.45
	X4	0	0	0	0	0	0.52	0	0.925
	X5	0.327	0.623	0.703	0.883	0.905	0.993	0.671	0.469
	X6	0.455	0.38	0.675	0.683	0.439	0.97	0.004	0.341
	X7	0	0.904	0.036	0.759	0.728	0.781	0	0.88
	X8	0	0	0	0.026	0	0.735	0	0.185
	X1*X2	0	0.027	0	0.815	0.799	0.912	0	0.558
	X1*X3	0	0.023	0.175	0.085	0.895	0.978	0	0.697
	X1*X4	0	0	0	0.13	0	0.834	0	0.754
	X1*X5	0.001	0.978	0.74	0.978	0.718	0.986	0.004	0.34
	X1*X6	0.001	0.948	0.711	0.923	0.996	0.952	0.691	0.325
	X1*X7	0	0.608	0.699	0.855	0.759	0.912	0	0.404
	X1*X8	0	0	0	0.02	0	0.937	0	0.006
Power Save	**X1**	0	0.887	0.04	0.764	0	0.322	0	0.885
	X2	0	0	0.007	0	0	0	0	0
	X3	0	0	0.148	0.918	0.003	0	0	0.472
	X4	0	0	0	0.002	0.619	0.731	0.001	0
	X5	0	0.926	0.023	0.868	0	0.169	0.022	0.803
	X6	0	0.576	0.085	0.876	0.033	0.968	0.53	0.001
	X7	0	0	0	0.149	0	0.652	0	0
	X8	0	0	0.024	0.214	0.096	0.281	0.191	0.085
	X1*X2	0	0.374	0	0.69	0.525	0.775	0	0.819
	X1*X3	0	0.065	0.302	0.142	0	0.966	0	0
	X1*X4	0.576	0.755	0.029	0.685	0	0.845	0.244	0.009
	X1*X5	0	0.084	0.03	0.766	0.535	0.168	0.08	0
	X1*X6	0.017	0.834	0.589	0.601	0.05	0.467	0.32	0.927
	X1*X7	0	0.519	0.33	0.994	0	0	0	0.504
	X1*X8	0	0.512	0.633	0.538	0.001	0.138	0.001	0
Random	**X1**	0.035	0.889	0.468	0.893	0.887	0.756	0	0.214
	X2	0	0	0	0	0	0	0	0
	X3	0	0.291	0.764	0.768	0.202	0.239	0	0.519
	X4	0	0	0.046	0.029	0	0.971	0	0.724
	X5	0.53	0.549	0.242	0.99	0.736	0.957	0.374	0.479
	X6	0.055	0.418	0.698	0.875	0.83	0.982	0	0.276
	X7	0	0.345	0.011	0.57	0.474	0.835	0	0.993
	X8	0	0	0.514	0.191	0.008	0.977	0	0.232
	X1*X2	0	0.427	0.332	0.873	0.697	0.89	0	0.42
	X1*X3	0	0	0.049	0.17	0.778	0.957	0	0.919
	X1*X4	0	0	0.795	0.76	0.001	0.621	0	0.401
	X1*X5	0.353	0.385	0.689	0.769	0.768	0.982	0	0.76
	X1*X6	0.039	0.94	0.412	0.988	0.652	0.98	0.556	0.474
	X1*X7	0	0.449	0.028	0.741	0.58	0.819	0	0.533
	X1*X8	0	0	0.034	0.599	0	0.427	0	0

Table 4.9 Examining the effects significance using p-value

It would be of interest, before proceeding to calculate the overall *WE* of the parameters and interactions, to examine under which *PA* the parameters and interactions achieved more significant effects on the co-residency metrics. There are 512 effect observations for the four *PA*s in Table 4.9, of which 225 were statistically significant (see the highlighted cells). Out of these significant effects, 28.88% were observed under First Fit, 23.55% under Next Fit, 26.66% under Power Save and 20.88% under Random.

In addition, it would also be of interest to see whether using two level ranges (i.e. used in the narrow-experiment and the broad-experiment) has shown any difference with regard to the effects significance. Out of all significant effects in Table 4.9, 72.88% were observed under the broad-experiment compared to 27.11% under the narrow-experiment.

4.5.2 Identifying the Most Influential Parameters on the Co-residency Metrics

Under each *PA*, the overall *WE* of each parameter/interaction were calculated as described in Section 4.3.4.3. As mentioned earlier, the maximum overall *WE* a parameter/interaction can achieve under each *PA* is eight, given that the parameter/interaction has achieved maximum *WE* (i.e. two) on the four co-residency metrics.

Table 4.10 outlines the overall *WE* of the parameters and interactions under First Fit, Next Fit, Power Save and Random *PA*s. In general, the 2-parameter interactions scored lower overall *WE* under the four *PA*s (average overall *WE* of 0.92) compared to the parameters (average overall *WE* of 1.99).

ID	Parameter/ Interaction	Overall Weighted Effect *WE*			
		Under First Fit	**Under Next Fit**	**Under Power Save**	**Under Random**
X1	Number of Clusters	1.064	1.216	0.990	0.245
X2	Number of Hosts	4.574	7.602	5.165	8.000
X3	Max Host Utilization	2.364	0.594	2.614	0.636
X4	Users' Arrival Rate	4.295	1.969	4.197	1.101
X5	Number of Users	0.967	0.000	0.802	0.000
X6	Parallel VMs per User	0.511	0.135	0.510	0.180
X7	VM Average Lifetime	3.665	0.609	3.563	0.565
X8	VMs per Request	1.816	1.682	1.435	0.618
X1*X2	Two-parameter interactions	0.715	1.376	0.839	0.239
X1*X3		2.142	0.456	1.716	0.656
X1*X4		1.109	1.689	0.801	0.742
X1*X5		0.476	0.150	0.501	0.175
X1*X6		0.442	0.017	0.017	0.013
X1*X7		2.000	0.580	1.851	0.721
X1*X8		1.209	2.160	1.492	1.517

Table 4.10 Overall *WE* of the parameters and interactions

The top four parameters and interactions with the highest overall *WE* under each *PA* are highlighted in Table 4.10. One major observation is that First Fit and Power Save share the same top four parameters and interactions, and the same observation applies for Next Fit and Random. Appendix C contains the parameters and interactions *WEs* on each of the co-residency metrics under each *PA*. Moreover, Appendix D presents the interaction plots of the significant 2-parameter interactions on the co-residency metrics.

Table 4.11 shows the parameters that scored the highest overall *WE* Under First Fit and Power Save:

Parameters	**Overall *WE* under First Fit**	**Overall *WE* under Power Save**
X2	4.573	5.165
X4	4.294	4.197
X7	3.664	3.563
X3	2.363	2.613

Table 4.11 The four parameters/interactions with the highest overall *WE* under First Fit and Power Save

For Next Fit and Random, Table 4.12 shows the parameters and interactions that scored the highest overall *WE*:

Parameters	Overall *WE* under Next Fit	Overall *WE* under Random
X2	7.601	8
X1*X8	2.160	1.516
X4	1.969	1.101
X1*X4	1.689	0.741

Table 4.12 The four parameters/interactions with the highest overall *WE* under Next Fit and Random

The Number of Hosts parameter (X2) repeatedly achieved the highest overall *WE* under the four *PA*s, achieving the maximum overall *WE* (i.e. 8) under Random. However, the parameters' overall *WEs* were not similar under different *PA*s. For instance, The Number of Hosts (X2) achieved a higher overall *WE* under Next Fit and Random (7.602 and 8.00)

compared to a relatively lower overall *WE* First Fit and Power Save (4.574 and 5.165). In contrast, The User Arrival Rate (X4) parameter had higher overall *WE* on First Fit and Power Save (4.295 and 4.197) compared to Next Fit and Random (1.969 and 1.101).

In the following figures, the overall *WE* results from Table 4.10 are illustrated to compare the parameters and interactions influence on the co-residency metrics under First Fit in (Figure 4.2), Next Fit in (Figure 4.3), Power Save in (Figure 4.4) and Random in (Figure 4.5).

The X-axis shows the parameters and the 2-parameter interactions and the Y-axis show their overall *WEs* on the co-residency metrics.

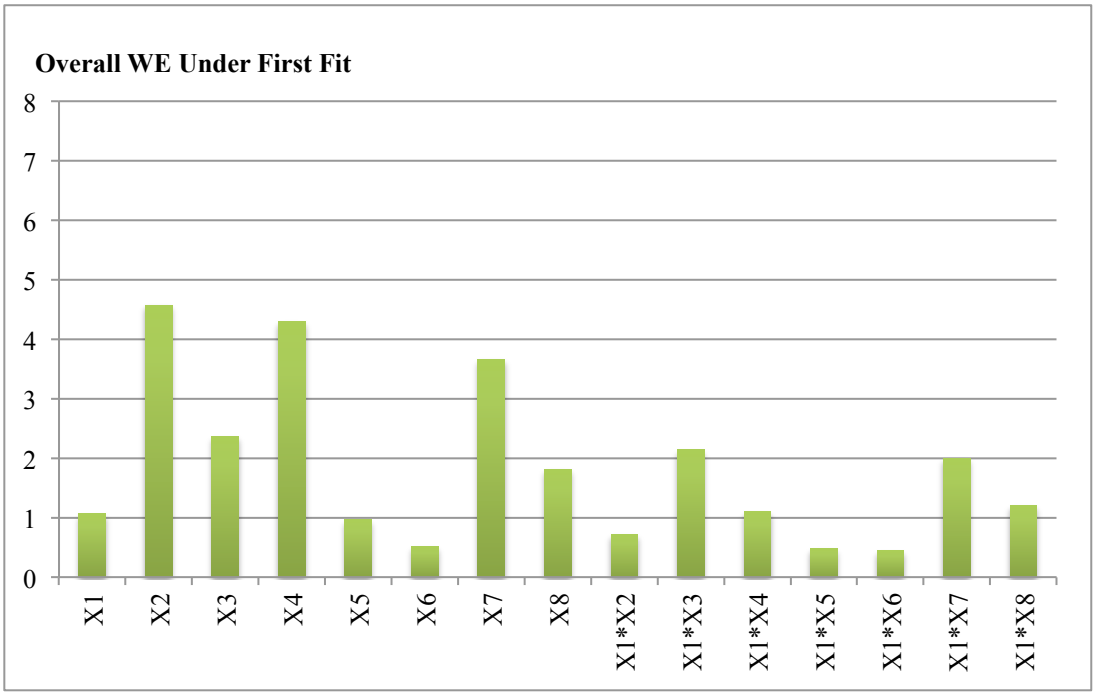


Figure 4.2 The overall Weighted Effect *WE* of the parameters/interactions under First Fit

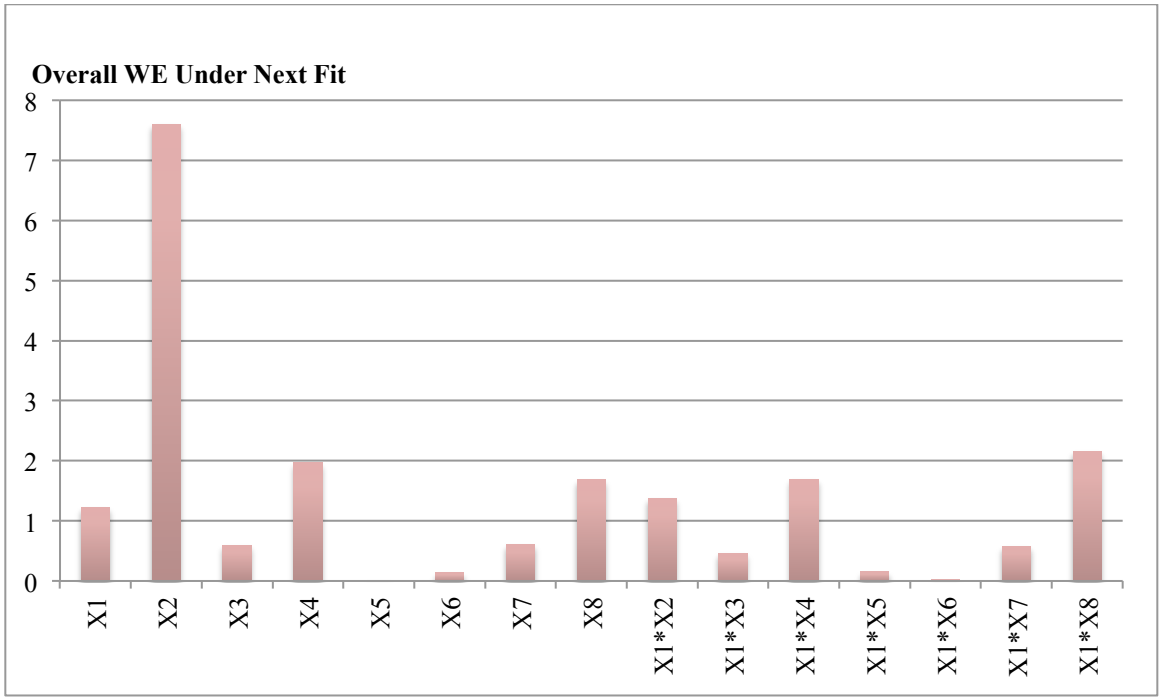


Figure 4.3 The overall Weighted Effect *WE* of the parameters/interactions under Next Fit

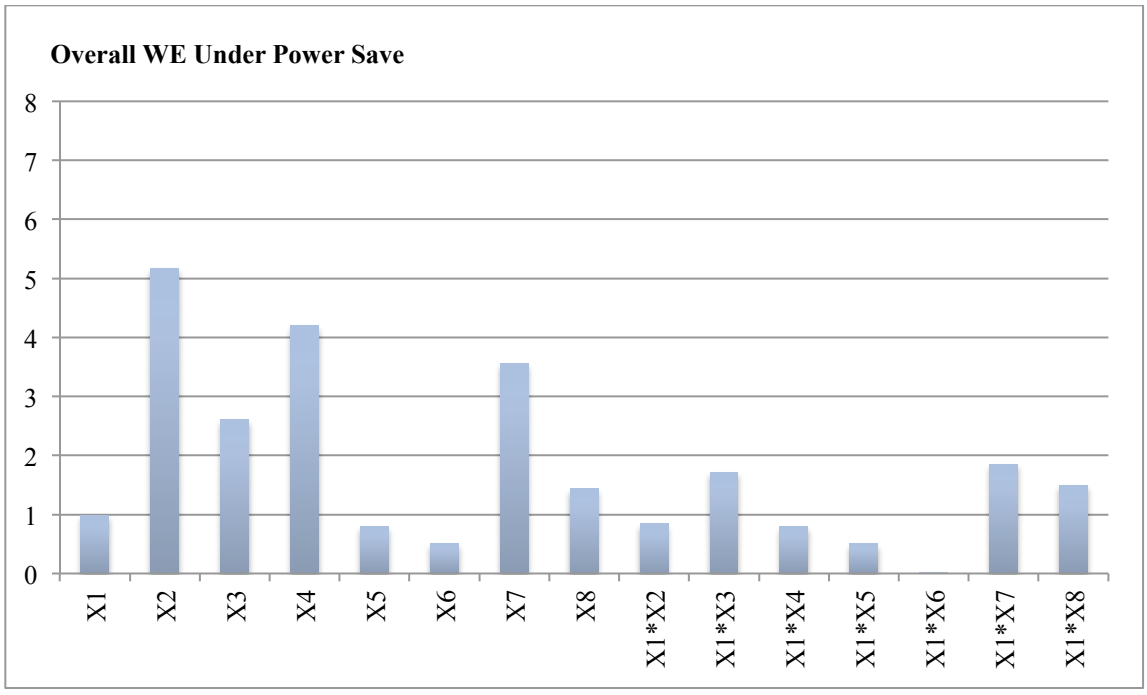


Figure 4.4 The overall Weighted Effect *WE* of the parameters/interactions under Power Save

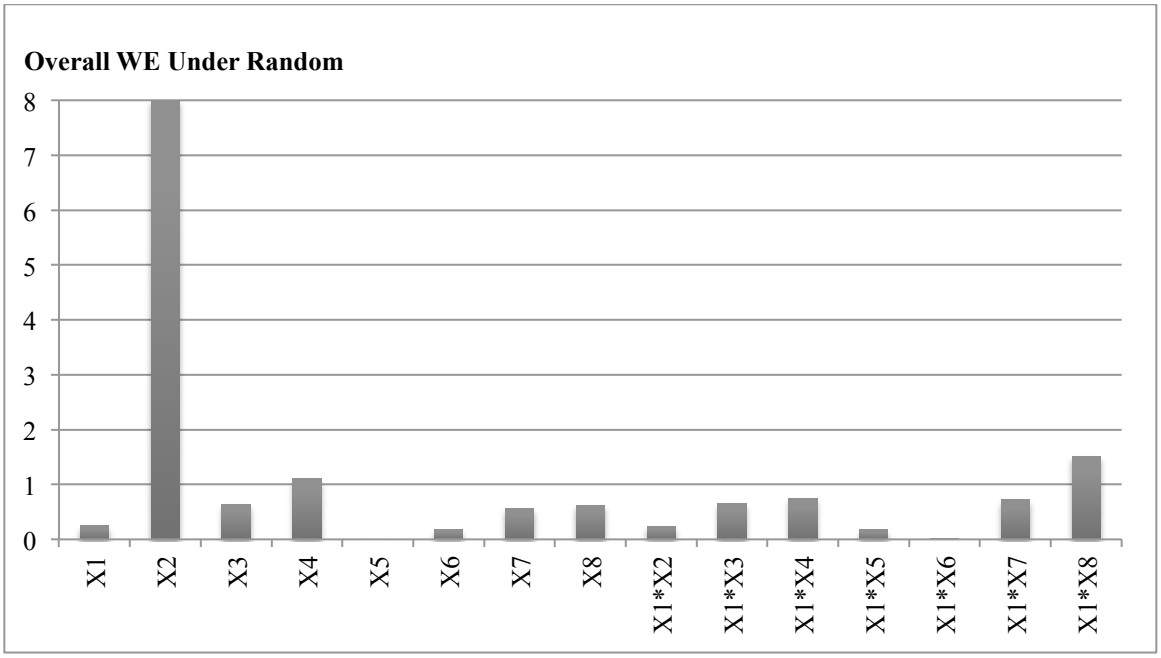


Figure 4.5 The overall Weighted Effect *WE* of the parameters/interactions under Random

4.6 Discussion

The main outcomes of this chapter are:

- **The four most influential parameters and interactions on the co-residency metrics are the same for First Fit and Power Save on one hand, and for Next Fit and Random, on the other hand.**

  This observation was the motivation for identifying the top four parameters and interactions that achieved the highest overall *WE* under a given *PA* as the most influential parameters and interactions on the co-residency metrics. This finding reveals that similarities exist between *PA*s in terms of what influences the co-residency behaviour in IaaS clouds (Table 4.11 and Table 4.12). The most likely cause for the similarity between First Fit and Power Save is that they share one common feature, that is, they prioritise the clusters and hosts with smaller IDs for new VMs placements (see Appendix A). On the other hand, one possible explanation of why Next Fit and Random have similar influential parameters and interactions is related to how clusters and hosts are selected for placement. In particular, clusters and hosts are selected in Next Fit in a cyclic fashion and in Random as a fair random

selection. For instance, for a given IaaS that has *C* clusters, Next Fit selects a given cluster with a probability of $\frac{1}{C}$. Similarly, the same cluster will be selected for a new VM placement in Random with the same probability. Appendix A describes in detail how the four *PA*s select clusters and hosts for VMs placement.

It is important to mention that *PA*s have been frequently compared for various applications such as [37], performance and energy consumptions [40], [55], [58], [99]. However, this thesis is the first to compare *PA*s in terms of their impact on co-residency behaviour. In addition, this thesis is the first to identify that a similarity exists between First Fit and Power Save, as well as between Next Fit and Random.

- **The four most influential parameters and interactions on the co-residency metrics are identified under First Fit, Next Fit, Power Save and Random.**

  This finding answers the second research question (i.e. For a given *PA*, what are the most influential cloud parameters affecting co-residency probability?). Table 4.10 shows that the quantified influence on the co-residency metrics varies between these parameters and interactions. This variation confirms the first research hypothesis that "for a given *PA*, cloud parameters such as the number of hosts and users do not have the same influence on the co-residency probability in IaaS clouds."

  Out of many parameters that define the IaaS cloud environment, one of the most significant findings in this chapter is that the number of hosts is the most influential parameter under four *PA*s. In addition, user arrival rate was identified among the four most influential parameters under four *PA*s.

  The following shows the four most influential parameters and two-parameter interactions under each *PA*:

  - Under First Fit and Power Save:

  The four most influential parameters on the co-residency metrics are (in order): Number of Hosts (X2), User Arrival Rate (X4), and VM Average Lifetime (X7) and Max Host Utilization (X3).

  - Under Next Fit and Random:

  The four most influential parameters on the co-residency metrics are (in order): Number of Hosts (X2), interaction of the Number of Clusters and the

VMs per Request parameters (X1*X8), User Arrival Rate (X4) and interaction of the Number of Clusters and the Users' Arrival Rate parameters (X1*X4). Unlike First Fit and Power Save, 2-parameter interactions were identified to be influential under Next Fit and Random.

This finding is particularly useful in this thesis to conduct further experiments in Chapter 5 on fewer, yet high influential, parameters. These experiments should provide valuable insights on what settings enable the influential parameters to reduce the probability of co-residency and under what *PA*.

- **When a 2-level experiment is used to quantify the parameters and interactions effects, the wider range between the two levels allows more significant effects compared to a narrower range to be observed.**
  The results in Table 4.9 suggest that the parameters are more likely to have significant effects on the co-residency metrics when they are varied between distant levels. This finding suggests that adding/removing a few clusters, for example, is less likely to cause a significant effect on the co-residency behaviour compared to a relatively larger change.
  In addition, this finding also suggests that using two different level ranges (narrow-range and broad-range) in the Influence Evaluation Strategy was useful to quantify the parameters influence on the co-residency metrics. The results show that the ratio of significant effects between the narrow-range and broad-range was approximately 1:3.

4.7 Summary

Perhaps the main challenge faced in this chapter is that there were many cloud parameters and parameters' settings to be included in limited resources experiments. In order to overcome this challenge, an Influence Evaluation Strategy has been proposed to simplify the process of designing experiments that have a large number of parameters and settings. The use of fractional factorial design was one step (of multiple steps) that the strategy applied to construct a reduced and balanced experiment. Using the *VMC* simulator as a testbed, this chapter has applied the Influence Evaluation Strategy to answer the second research question on what parameters influence the co-residency metrics the most.

Under each of the used *PA*s in this thesis, the strategy was able to quantify the influence of eight cloud parameters and their interactions on the co-residency metrics. This quantification led to identifying the four most influential parameters and 2-parameter interactions on the co-residency metrics (Table 4.11 and Table 4.12). One of the most important findings in this chapter is that the number of hosts is the most influential parameter under four *PA*s. Under First Fit and Power Save, the four most influential parameters on the co-residency metrics were number of hosts, user arrival rate, VM average lifetime and maximum host utilization. On the other hand, the four most influential parameters under Next Fit and Random were the number of hosts, the interaction of the number of clusters and VMs per request parameters, user arrival rate and the interaction of the number of clusters and users' arrival rate parameters.

In addition, this thesis is the first to compare these four *PA*s in terms of their impact on the co-residency metrics. The findings show that a similarity exists between First Fit and Power Save, as well as between Next Fit and Random.

Further, the results presented in Table 4.10 support the first hypothesis put forward in Section 1.3. The first hypothesis states "for a given *PA*, cloud parameters such as the number of hosts and users do not have the same influence on the co-residency probability in IaaS clouds."

The next chapter (Chapter 5) is dedicated to answering the third research question on how the most influential parameters' settings can be used to reduce the co-residency probability in four *PA*s.

Chapter 5
Reducing Co-residency Probability

5.1 Introduction

This chapter aims to answer the third research question of "what are the parameter settings that reduce the co-residency probability in a given *PA*." The co-residency probability determines the chance that a VM experiences an arbitrary co-residency hit (see Section 1.2). Chapter 4 identified the most influential parameters and 2-paremeter interactions on the co-residency metrics (Table 4.11 and Table 4.12). Under First Fit, Next Fit, Power Save and Random, this chapter employs the *VMC* simulator to estimate the co-residency metrics using controlled experiments. These estimates are obtained by examining the influential parameter at more levels under four *PA*s.

This approach serves two important functions. First, these estimates are used to test the influential parameters at more levels to investigate the relationship between each parameter and the co-residency metrics, in order to determine the best parameter settings that reduce the co-residency probability. For instance, does the increase in the number of hosts reflect a linear increase or decrease in the co-residency probability? Second, comparing *PA*s in terms of reducing the co-residency probability.

Since that the co-residency metrics characterize probabilities related to co-residency, an assumption is made in this chapter that the co-residency probability is reduced when:

- The Co-residency Coverage Probability *CCP* is **reduced**.
- The Hit-Free Lifetime *HFL* is **increased**.
- The Co-residency Vacancy *CV* is **reduced**.
- The Co-residency Activity *CA* is **reduced**.

The remainder of this chapter is organized as follows. The next section describes the method and the experiment settings that were used to estimate the co-residency metrics. The main conclusions are presented in Section 5.3 and discussed in Section 5.4.

5.2 Method

The *VMC* simulator was used to estimate the co-residency metrics under four *PA*s (i.e. First Fit, Next Fit, Power Save and Random). This section defines how controlled experiments were conducted to examine further the influential parameters (identified in Chapter 4) using ten levels listed in Table 5.1. A controlled experiment is one in which all parameters are held constant except for one [14]. This section also explains how the results were analyzed to answer the third research question on which parameters' settings reduce the co-residency probability in four *PA*s.

5.2.1 Experimental Setup

Chapter 4 identified the four most influential parameters under four *PA*s (Table 4.11 and Table 4.12). Therefore, one controlled experiment per influential parameter was conducted under each *PA*. In each experiment, the same eight parameters from Chapter 4 (Table 5.2) were separated into two groups: an experimental group and a control group. The experimental group contained one influential parameter that was tested at ten levels, while the control group consisted of the remaining 7 parameters that were kept constant.

With ten new parameter levels defined in Table 5.1, each controlled experiment consisted of ten experimental runs. These new levels were selected such that they were evenly distributed between the low level and the high level of the broad-range (Table 4.4). This action ensured more levels covered, especially the levels that were not tested by the Influence Evaluation Strategy in Chapter 4.

In addition, Chapter 4 identified two 2-parameters interactions (i.e. X1*X4 and X1*X8) that had an influence under Next Fit and Random. The use of controlled experiments allowed testing each of the interacting parameters individually at ten levels while keeping the other control parameters (including the interacting parameter) at a constant level.

There were many possible levels that could be assigned to the control parameters. Appendix D illustrates the significant two 2-parameters interactions (i.e. X1*X4 and X1*X8) and reveals that nearly 63.6% of these interactions were able to reduce the co-residency probability when both X4 and X8 were in low levels. This finding was one of the motivations for assigning the low levels from the narrow-range (Table 4.4) to the control parameters.

New levels of the most Influential Parameters					
Number of Clusters (X1)	**Number of Hosts (X2)**	**Max Host Utilization (X3)**	**Users' Arrival Rate (X4)**	**VM Average Lifetime (X7)**	**VMs per Request (X8)**
15	1000	80%	2	2000	2
19	4000	82%	2.33	2150	2.2
23	7000	84%	2.66	2350	2.4
29	10000	85%	2.99	2550	2.6
33	13000	88%	3.33	2700	2.8
37	16000	91%	3.66	2850	3
41	19000	94%	3.99	3050	3.2
44	22000	96%	4.33	3250	3.5
47	25000	98%	4.66	3450	3.8
50	30000	100%	5	3600	4

Table 5.1 New levels for testing the most influential parameters.

For each influential parameter under a given *PA*, a controlled experiment was conducted using the following steps:

1. The remaining parameters (i.e. the control group) were kept constant. Table 5.2 lists the levels that were used to fix the control parameters.

Parameter	Control level
Number of Clusters (X1)	15
Number of Hosts (X2)	1000
Max Host Utilization (X3)	80%
Users' Arrival Rate (X4)	2
Number of Users (X5)	35000
Parallel VMs per User (X6)	12
VM Average Lifetime (X7)	2000
VMs per Request (X8)	2

Table 5.2 Control level for parameters

2. The influential parameter's levels from Table 5.1 were used in the *VMC* simulator while holding the control group parameters constant.
3. To increase the reliability of the experiment's results, each of the ten influential parameters' levels was tested in ten simulation repetitions. This provided 100 observations per parameter per *PA*.

5.2.2 Analysis Approach

With four influential parameters per *PA* and ten levels per parameters (tested in ten simulation repetitions), each *PA* was examined in 400 simulation runs yielding a total of 1600 simulation runs under four *PA*s. These simulations' results showed the following under each *PA*:

1. Simulation estimates of the co-residency metrics (used to compare *PA*s in terms of reducing the co-residency probability).
2. The correlation between the influential parameters and the co-residency metrics (used to identify the best parameter settings that reduce the co-residency probability).

The co-residency metrics estimates were used to identify the best parameters' settings that reduced the co-residency probability in four *PA*s. Each of the co-residency metrics was estimated with 99 degrees of freedom under each *PA* to increase the estimates accuracy. Confidence Intervals with 95% confidence level were used to enhance the precision of these

estimates. These intervals describe the likely range of a sample estimate from the true population. Confidence intervals are reported in tables as (Mean ± margin of error). However, it is important to note that outliers (i.e. observation points that are distant from other observations) can have an impact on the confidence interval [83]. The sample Pearson correlation coefficient was used to describe the linear correlation between parameters and metrics to obtain reliable estimates of the co-residency metrics and reduce the effect of these outliers. Pearson coefficients are sensitive to outliers, and the strongest correlations (i.e. 1.0 and -1.0) occur when data points fall exactly on a straight line. In this thesis, the stronger the Pearson's correlation coefficients, the better the estimate.

More importantly, calculating the Pearson's correlation coefficients between the influential parameters and the co-residency metrics revealed valuable insights that helped to identify the best parameters' settings that reduced the co-residency probability under each *PA*.

The method in which the correlations were obtained and interpreted is described in the following section (Section 5.2.3).

5.2.3 Influential Parameters Correlations with the Co-residency Metrics

The sample Pearson correlation coefficient, or the r-value for short, was used to examine the influential parameters linear correlations with the co-residency metrics. The r-value can be any value between +1 and −1, where +1 indicates a total positive correlation, 0 indicates no correlation, and −1 indicates a total negative correlation [14]. As pointed out in the previous section, the strongest correlations (i.e. 1.0 and -1.0) occur when data points fall exactly on a straight line. The r-value is also useful to indicate the slope of the correlation, where a positive r-value indicates that an increase on the influential parameter's level results in an increase on the corresponding co-residency metric and vice versa.

Dancey and Reidy suggested the following categorisation of the strength of correlation as shown in Table 5.3 [26]:

|r-value|	Strength of Correlation
1	Perfect
0.7 - 0.9	Strong
0.4 - 0.6	Moderate
0.1 - 0.3	Weak
0	Zero

Table 5.3 Categorisation of the strength of correlation

In this chapter, an influential parameter was considered to have a strong linear correlation with a given co-residency metric if the corresponding |r-values| is between 0.4 to1.0. Figure 5.1 illustrates three examples of a moderate positive correlation at r-value = 0.5, no correlation at r-value = 0 and a strong negative correlation at r-value = -1.0.

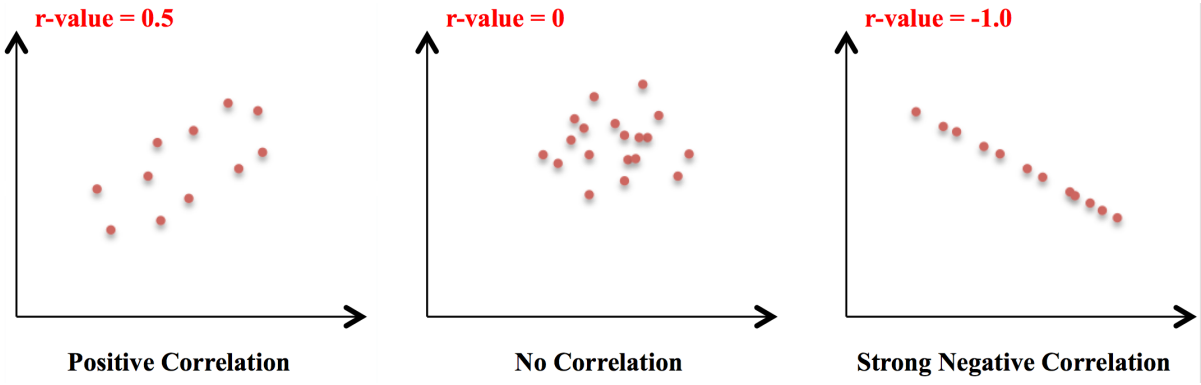


Figure 5.1 Examples of correlation r-values.

Given that n is the number of an influential parameter's observations in a controlled experiment (i.e. 100 in this chapter), $\bar{A}$ is the average of the parameter's levels and $\bar{M}$ is the co-residency metric's estimate, the sample Pearson correlation coefficient, r-value, was calculated as follows:

$$r - value = \frac{\sum_{i=1}^{n}[(A_i - \bar{A})(M_i - \bar{M})]}{\sqrt{\sum_{i=1}^{n}(A_i - \bar{A})^2}\sqrt{\sum_{i=1}^{n}(M_i - \bar{M})^2}}$$

Finding the r-value between the influential parameters and the co-residency metrics served two important purposes:

1. To verify whether the limitation identified in Section 4.3.2 had any effects on the Influence Evaluation Strategy outcomes. This limitation was present because the strategy measured each parameter's effects between two levels only. Therefore, there is no guarantee that the parameter held a strong linear effect between these two levels, that is, no outlier was present. In contrast, the controlled experiments in this chapter tested more levels. Therefore, a weak r-value (i.e. $|\text{r-value}| < 0.4$, see Table 5.3) between an influential parameter and the co-residency metrics can indicate the presence of outliers that were not detected by the Influence Evaluation Strategy.
2. The r-value indicates the slope of the correlation between an influential parameter and the co-residency metrics. Therefore, it was used to identify the best parameters' settings that reduced the co-residency probability under each *PA*. For instance, does the increase in the number of hosts reflect a linear increase or decrease in the co-residency probability?

Having defined the method that was used in this chapter, the following section highlights key findings from the controlled experiments.

5.3 Findings

This section outlines the important findings concerning the best parameters' settings at which the co-residency probability was reduced in four *PA*s. Since the co-residency metrics estimate probabilities related to co-residency, an assumption was made that the co-residency probability can be reduced by:

- **Reducing** the Co-residency Coverage Probability *CCP*,
- **Increasing** the Hit-Free Lifetime *HFL*,
- **Reducing** the Co-residency Vacancy *CV* and
- **Reducing** the Co-residency Activity *CA*.

The findings in the following sections can best be treated under the previous four headings. The following (Table 5.4, Table 5.5, Table 5.6 and Table 5.7) summarize the maximum and minimum observed values of the co-residency metrics under First Fit, Next Fit, Power Save and Random. In addition, the tables show the r-values of the correlation between the metric

and each of the influential parameters. The empty cells under each *PA* indicate that the corresponding parameter was not identified as an influential parameter under that particular *PA*.

	Parameters	First Fit			Next Fit			Power Save			Random		
		Min	Max	r-value	Min	Max	r-value	Min	Max	r-value	Min	Max	r-value
***CCP* Metric**	**X1**				0.929	0.939	0.057				0.925	0.937	0.074
	X2	0.834	0.891	-0.396	0.000	0.937	-0.975	0.822	0.882	-0.175	0.240	0.935	-0.954
	X3	0.819	0.915	0.428				0.820	0.913	-0.026			
	X4	0.749	0.891	-0.575	0.923	0.936	-0.492	0.762	0.890	-0.677	0.919	0.936	-0.784
	X7	0.734	0.885	-0.891				0.732	0.882	-0.889			
	X8				0.930	0.940	0.444				0.929	0.937	0.136

Table 5.4 The r-values, minimum and maximum *CCP* observed under each *PA*

	Parameters	First Fit			Next Fit			Power Save			Random		
		Min	Max	r-value	Min	Max	r-value	Min	Max	r-value	Min	Max	r-value
***HFL* Metric**	**X1**				0.142	0.161	-0.159				0.132	0.148	-0.281
	X2	0.099	0.137	-0.589	0.148	1.00	0.883	0.097	0.130	-0.354	0.136	0.500	0.801
	X3	0.104	0.156	-0.434				0.102	0.151	-0.315			
	X4	0.097	0.216	0.523	0.095	0.157	-0.877	0.099	0.235	0.630	0.098	0.150	-0.806
	X7	0.033	0.149	-0.870				0.030	0.143	-0.891			
	X8				0.106	0.158	-0.900				0.101	0.147	-0.901

Table 5.5 The r-values, minimum and maximum *HFL* observed under each *PA*

	Parameters	First Fit			Next Fit			Power Save			Random		
		Min	**Max**	**r-value**	**Min**	**Max**	**r-value**	**Min**	**Max**	**r-value**	**Min**	**Max**	**r-value**
***CV* Metric**	**X1**				0.1096	0.1547	-0.288				0.0766	0.1098	-0.152
	X2	0.0005	0.0071	-0.690	0.421	0.983	0.538	0.0003	0.0057	-0.727	0.242	0.977	0.548
	X3	0.0008	0.2202	0.549				0.0009	0.2126	-0.087			
	X4	0.0003	0.0038	-0.281	0.0425	0.1472	-0.869	0.0003	0.0048	-0.274	0.0339	0.1137	-0.898
	X7	0.0003	0.0049	-0.264				0.0003	0.0054	-0.362			
	X8				0.0776	0.1530	-0.857				0.0578	0.0992	-0.877

Table 5.6 The r-values, minimum and maximum *CV* observed under each *PA*

	Parameters	First Fit			Next Fit			Power Save			Random		
		Min	**Max**	**r-value**	**Min**	**Max**	**r-value**	**Min**	**Max**	**r-value**	**Min**	**Max**	**r-value**
***CA* Metric**	**X1**				0.0027	0.0034	0.192				0.0024	0.0033	0.163
	X2	0.0001	0.0006	-0.868	0.0000	0.0032	-0.732	0.0001	0.0006	-0.880	0.0001	0.0030	-0.792
	X3	0.0004	0.0017	0.695				0.0005	0.0018	0.749			
	X4	0.0001	0.0006	-0.870	0.0018	0.0032	-0.810	0.0001	0.0006	-0.844	0.0016	0.0031	-0.848
	X7	0.0001	0.0015	-0.504				0.0001	0.0020	-0.494			
	X8				0.0026	0.0031	-0.573				0.0022	0.0029	-0.666

Table 5.7 The r-values, minimum and maximum *CA* observed under each *PA*

5.3.1 Reducing the Co-residency Coverage Probability (*CCP)*

This section describes the findings concerning the Co-residency Coverage Probability *CCP*. The aim is to identify the best parameters' settings at which the *CCP* estimate was low in four *PA*s.

	CCP Estimate	± 95.0% confidence interval of the estimate	r-value
First Fit	0.845	0.0018	-0.396
Next Fit	0.379	0.065	-0.975
Power Save	0.840	0.0019	-0.175
Random	0.501	0.043	-0.954

Table 5.8 The *CCP* estimate with Number of Hosts (X2) varying between 1000-30000

Table 5.8 shows a relatively high amount of variability in the *CCP* estimates. This variation suggested that changing the number of hosts caused greater variability in the *CCP*, and the degree of this variability is different between the *PA*s and higher in Next Fit and Random. Figure 5.2 reveals that there has been a steep decline in the *CCP* when the Number of Hosts (X2) has been increased using Next Fit. For instance, the figure shows that the *CCP* reached zero for IaaS clouds with a number of hosts larger than 25000. Similarly, the use of Random reduced the *CCP* to 0.240 with a very strong negative correlation. The same figure also shows that the Number of Hosts (X2) had negative correlations with the *CCP* under all *PA*s. This negative correlation indicates that increasing the number of hosts in an IaaS cloud contributed to reducing the probability of co-residency. However, the negative correlation is higher in Next Fit and Random as seen in the sharp drop of the *CCP* compared to the slight decrease of the *CCP* under First Fit and Power Save.

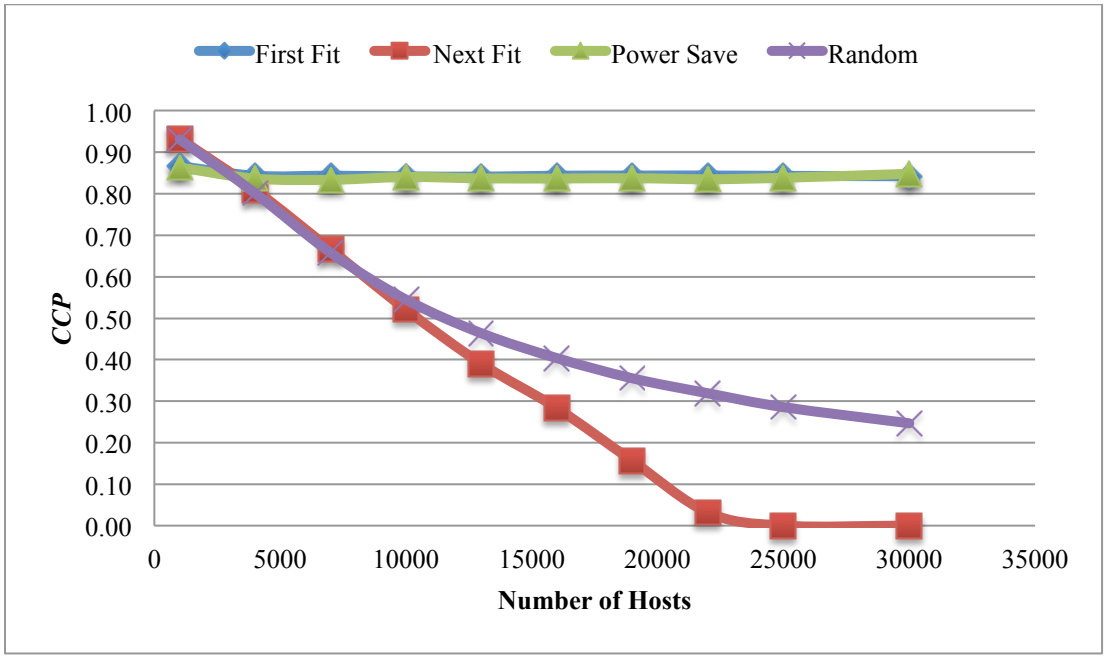


Figure 5.2 The *CCP* metric at different Number of Hosts (X2)

5.3.2 Increasing the Hit-Free Lifetime (*HFL*)

This section describes the findings concerning the Hit-Free Lifetime *HFL* metric. The aim is to identify the best parameters' settings at which the *HFL* estimate was high in four *PA*s.

	***HFL* Estimate**	**± 95.0% confidence interval of the estimate**	**r-value**
First Fit	0.108	0.0012	-0.589
Next Fit	0.771	0.054	0.883
Power Save	0.108	0.0011	-0.354
Random	0.414	0.020	0.801

Table 5.9 The *HFL* estimates under different Number of Hosts (X2) ranging between 1000-30000

This variation in the *HFL* estimates between *PA*s (Table 5.9) suggested that changing the number of hosts caused greater variability in the *HFL*, and the degree of this variability is different between the *PA*s and higher in Next Fit and Random.

Figure 5.3 reveals that there has been a gradual increase in the lifetime ratio at which a VM is safe from co-residency hits (i.e. *HFL*) when the Number of Hosts (X2) was increased using Next Fit. The figure shows that the *HFL* reached a peak value of 1.00 (i.e. the entire lifetime of a given VM was hit-free). In addition, the use of Random prolonged the *HFL* to 0.414. Similar to the Co-residency Coverage Probability (Section 5.3.1), Next Fit and Random showed to have strong positive correlations that can be seen in the gradual rise of the *HFL*. In contrast, the same figure shows that the number of hosts had relatively weaker negative correlations that can be seen in the steady decline of the *HFL* under First Fit and Power Save.

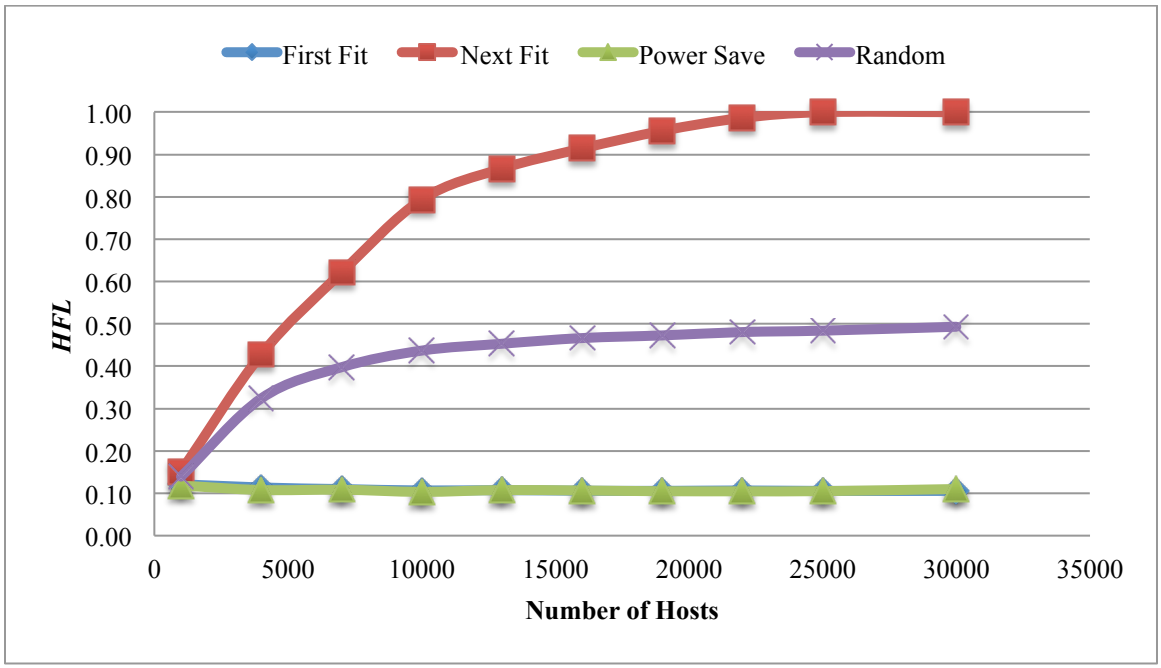


Figure 5.3 The *HFL* metric at different Number of Hosts (X2)

In addition, as the Users' Arrival Rate (X4) increased, the lifetime ratio at which a VM is safe from co-residency hits (i.e. *HFL)* has increased, showing a positive correlation using First Fit and Power Save. The Hit-Free Lifetime *HFL* estimates under different *PA*s and different users' arrival rates ranging between 2-5 are shown in Table 5.10:

	HFL Estimate	± 95.0% confidence interval of the estimate	r-value
First Fit	0.146	0.0056	0.523
Next Fit	0.120	0.0035	-0.877
Power Save	0.139	0.0045	0.630
Random	0.116	0.0027	-0.806

Table 5.10 The *HFL* estimates with Users' Arrival Rates (X4) varying between 2-5

Table 5.10 shows high variations in the *HFL* estimates. These variations indicated that varying the users' arrival rate caused a smaller amount of change on the *HFL* compared to varying the number of hosts on the *HFL* (Table 5.9).

Figure 5.4 reveals that there has been a slight increase in the *HFL* when the users' arrival rate has been increased under First Fit. The figure shows that the *HFL* reached a peak value of 0.216 (i.e. approximately 21.6% of a given VM lifetime was hit-free) with a moderate positive correlation. Similarly, the use of Power Save extended the *HFL* to 0.235 with a strong positive correlation. In contrast, the same figure shows that the users' arrival rate had relatively stronger negative correlations with the *HFL* under Next Fit and Random.

Figure 5.4 The *HFL* metric at different Users' Arrival Rates (X4)

5.3.3 Reducing the Co-residency Vacancy *(CV)*

This section describes the findings concerning the Co-residency Vacancy *CV*. The aim is to identify the best parameters' settings at which the *CV* estimate was low in four *PA*s.

The *CV* estimates under different *PA*s, and different number of hosts ranging between 1000-30000 are shown in Table 5.11:

	***CV* Estimate**	**± 95.0% confidence interval of the estimate**	**r-value**
First Fit	0.0018	0.0002	-0.690
Next Fit	0.922	0.125	0.538
Power Save	0.0014	0.0002	-0.727
Random	0.892	0.163	-0.690

Table 5.11 The *CV* estimates with Number of Hosts (X2) varying between 1000-30000

Table 5.11 reveals a relatively high variation in the *CV* estimates between *PA*s. Figure 5.5 reveals that increasing the Number of Hosts (X2) from 1000 to 7000 hosts under Next Fit and Random caused a sharp rise of the *CV*. For instance, the *CV* reached the maximum observed values of 0.9667 and 0.9095 respectively with moderate positive correlation. As the number of hosts exceeds 7000, the observed values of the *CV* under Next Fit and Random were clustered toward the maximum possible value of 1 (see Section 3.4.3). In contrast, Figure 5.5 shows that there has been a smaller change in the *CV* when the number of hosts has been increased using both First Fit and Power Save. For example, the *CV* reached a low value of 0.0005 (i.e. a given VM's host has been available for VMs placement during 0.05% of the VM lifetime) with a strong negative correlation.

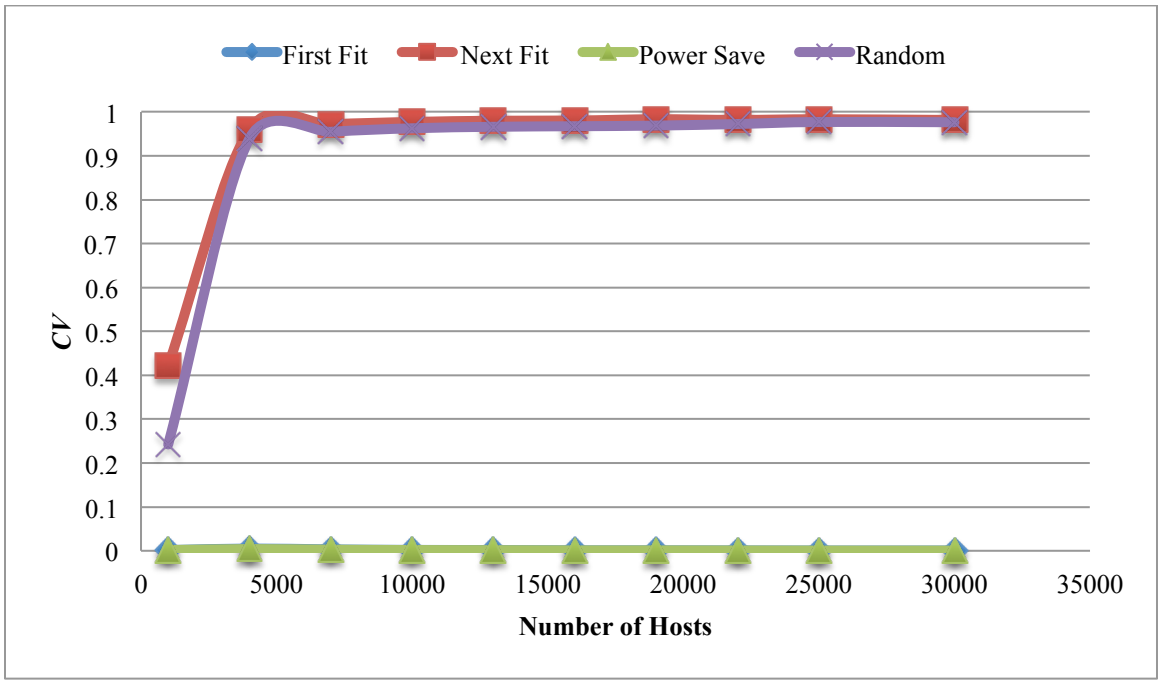


Figure 5.5 The *CV* metric at different Number of Hosts (X2)

In addition, as the Users' Arrival Rate (X4) increased, the *CV* has decreased under all *PA*s (i.e. negative correlations). The Co-residency Vacancy *CV* estimates under different *PA*s and different users' arrival rates ranging between 2-5 are shown in Table 5.12:

	CV Estimate	± 95.0% confidence interval of the estimates	r-value
First Fit	0.0013	0.0001	-0.281
Next Fit	0.0845	0.0052	-0.869
Power Save	0.0011	0.0001	-0.274
Random	0.0602	0.0042	-0.898

Table 5.12 The *CV* estimates with Users' Arrival Rates (X4) varying between 2-5

One interesting observation from Table 5.12 is that varying the users' arrival rate caused a smaller amount of change on the *CV* estimates compared to varying the number of hosts (Table 5.11).

Figure 5.6 reveals that the *CV* reached a low value of 0.0003 using First Fit and Power Save. In contrast, the same figure shows that the users' arrival rate had far stronger negative correlations with the *CV* under Next Fit and Random. However, reducing the *CV* was better achieved under First Fit and Power Save when the users' arrival rate varies.

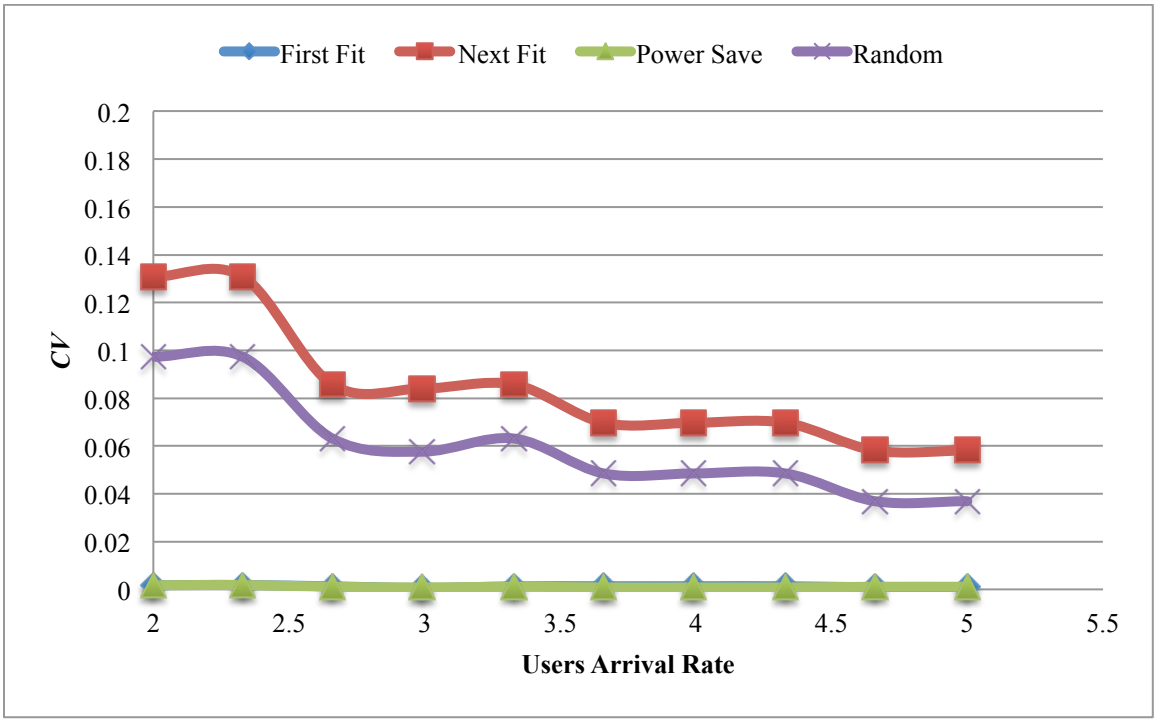


Figure 5.6 The *CV* metric at different Users' Arrival Rates (X4)

In addition, Table 5.13 shows the Co-residency Vacancy *CV* estimates under different VMs' average lifetime ranging between 2000-3600 in First Fit and Power Save. The VMs average lifetime parameter (X7) was identified to be among the most influential parameters on the co-residency metrics under First Fit and Power Save. Further, Figure 5.7 shows that the *CV* was kept at lower values that reached 0.0003 under different VMs average lifetime values.

	***CV* Estimate**	**± 95.0% confidence interval of the estimate**	**r-value**
First Fit	0.0019	0.0001	-0.264
Power Save	0.0019	0.0001	-0.362

Table 5.13 The *CV* estimates with VMs Average Lifetime (X7) varying between 2000-3600

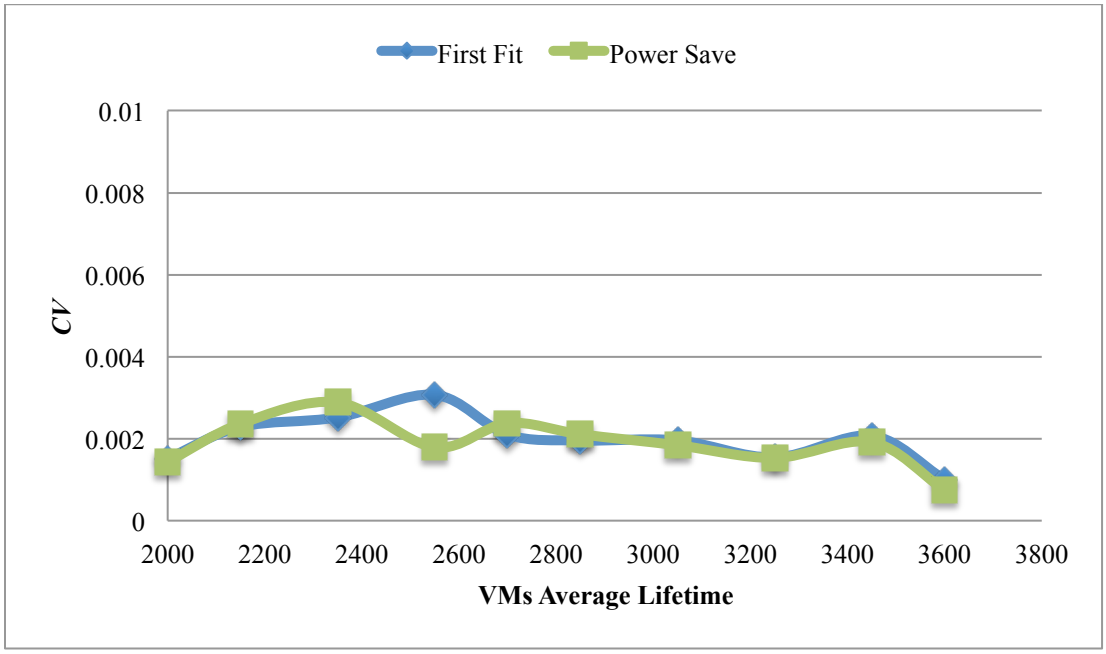


Figure 5.7 The *CV* metric at different VMs Average Lifetime (X7)

5.3.4 Reducing the Co-residency Activity (*CA)*

This section describes the findings concerning the Co-residency Activity *CA*. The aim is to identify the best parameters' settings at which the *CA* estimate was low in four *PA*s.

The Co-residency Activity *CA* estimates under different *PA*s and different number of hosts ranging between 1000-30000 are shown in Table 5.14:

	CA Estimate	± 95.0% confidence interval of the estimate	r-value
First Fit	0.0002	$3x10^{-5}$	-0.868
Next Fit	0.0005	0.0001	-0.732
Power Save	0.0002	$3x10^{-5}$	-0.880
Random	0.0006	0.0001	-0.792

Table 5.14 The *CA* estimates with Number of Hosts (X2) varying between 1000-30000

As the Number of Hosts (X2) increased, the Co-residency Activity *CA* has decreased under all *PA*s with strong negative correlations. Figure 5.8 reveals that the *CA* was minuscule regardless of the number of hosts or the *PA*, and reached the lowest value (i.e. zero) using Next Fit, and 0.0001 in the remaining *PA*s.

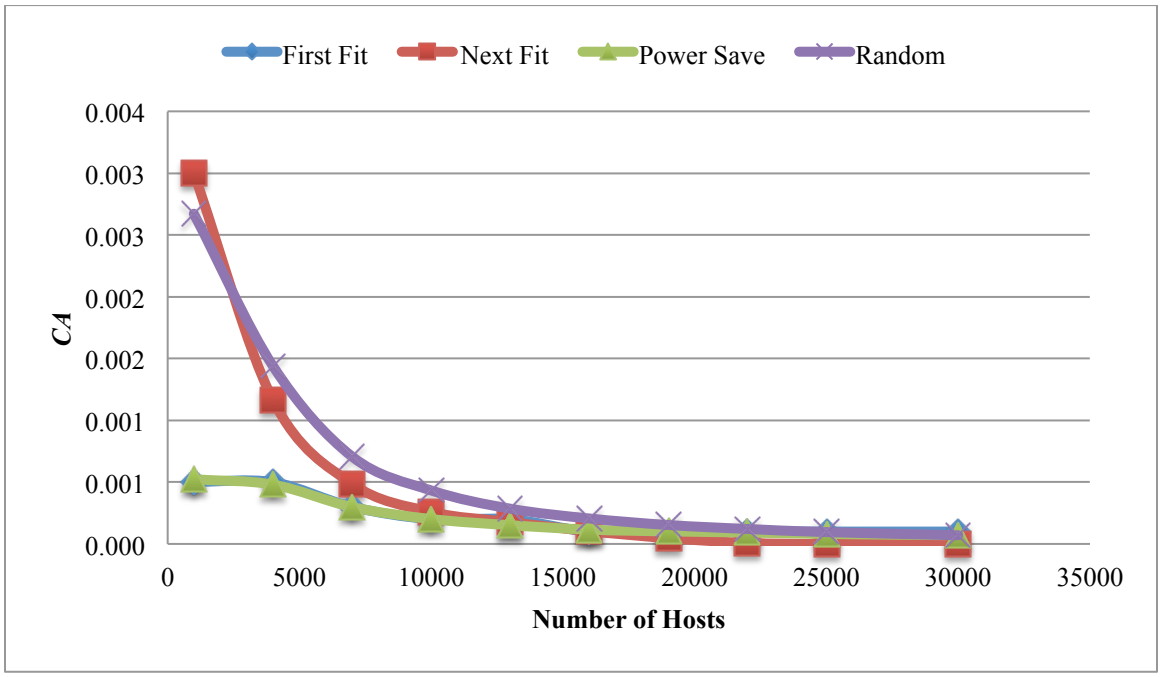


Figure 5.8 The *CA* metric at different Number of Hosts (X2)

In addition, as the Users' Arrival Rate (X4) increased, the *CA* has decreased under all *PA*s. The Co-residency Activity *CA* estimates under different *PA*s and different users' arrival rate ranging between 2-5 are shown in Table 5.15:

	***CA* Estimate**	**± 95.0% confidence interval of the estimate**	**r-value**
First Fit	0.0003	$2x10^{-5}$	-0.870
Next Fit	0.0024	$6x10^{-5}$	-0.810
Power Save	0.0002	$2x10^{-5}$	-0.844
Random	0.0021	$7x10^{-5}$	-0.848

Table 5.15 The *CA* estimates with Users' Arrival Rates (X4) varying between 2-5

Increasing the users' arrival rate had a negative correlation with the Co-residency Activity *CA* and the Co-residency Vacancy *CV* (Section 5.3.3) under all *PA*s. Figure 5.9 reveals that the *CA* reached a low value of 0.0001 using First Fit and Power Save.

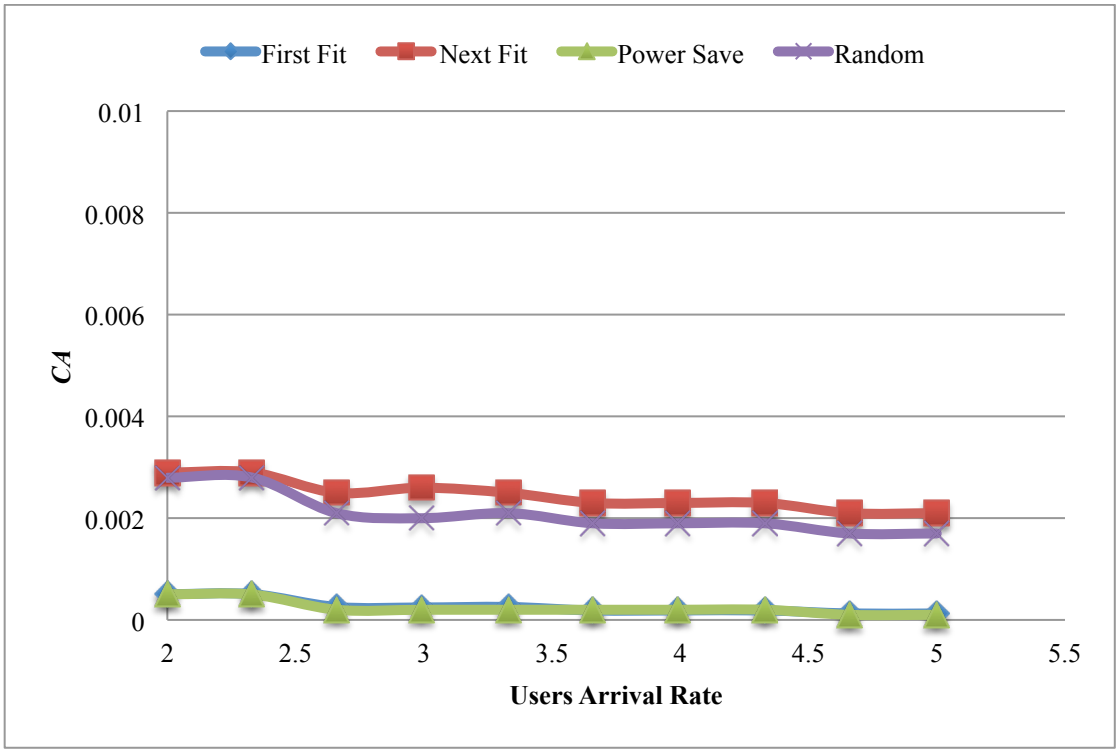


Figure 5.9 The *CA* metric at different Users' Arrival Rate (X4)

With respect to the VMs average lifetime (X7), Figure 5.10 shows that the *CA* was kept at lower values as the VMs average lifetime increased in the First Fit as well as Power Save, reaching a low value of 0.0001.

Further, the Co-residency Activity *CA* estimates under different VMs' average lifetime ranging between 2000-3600 in First Fit and Power Save are shown in Table 5.16:

	***CA* Estimate**	**± 95.0% confidence interval of the estimate**	**r-value**
First Fit	0.0007	$6x10^{-5}$	-0.504
Power Save	0.0007	$7x10^{-5}$	-0.494

Table 5.16 The *CA* estimates with VMs Average Lifetime (X7) varying between 2000-3600

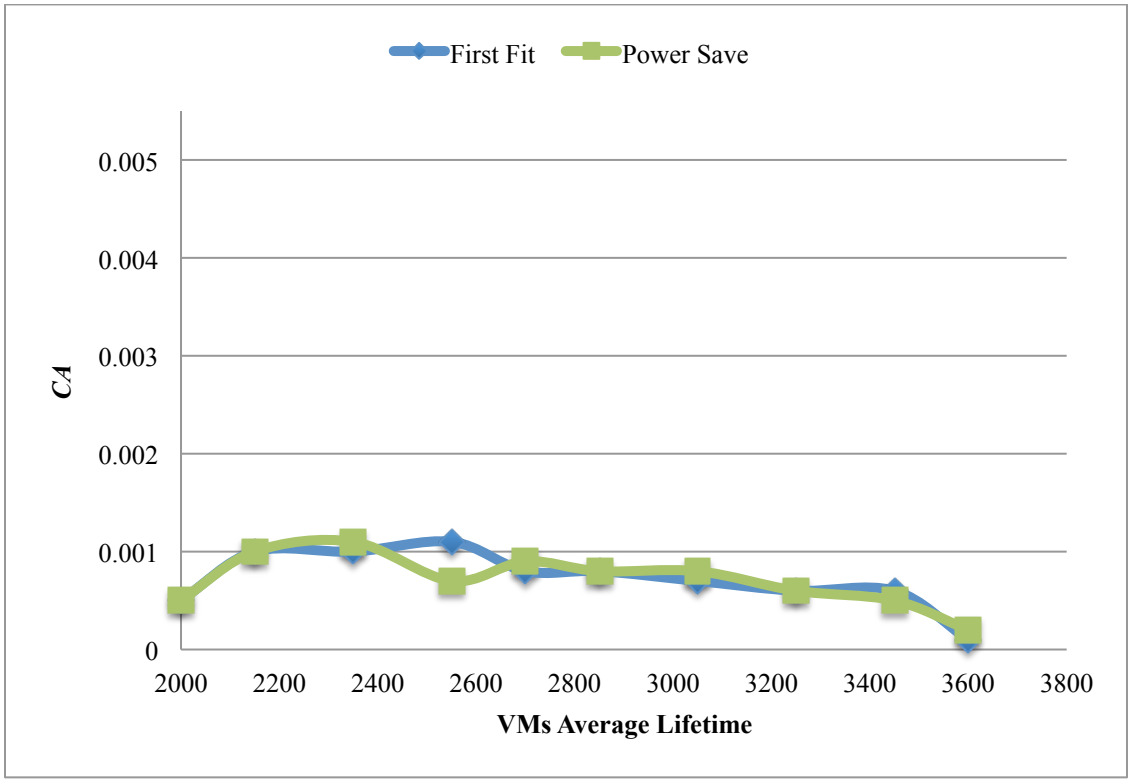


Figure 5.10 The *CA* metric at different VMs Average Lifetime (X7)

Before proceeding to discuss the previous findings and their practical uses on reducing the co-residency probability in IaaS clouds, the next section provides a summary of the influential parameters linearity check results. These results will be used in Section 5.4 to evaluate the effectiveness and efficiency of the Influence Evaluation Strategy in identifying the most influential parameters on the co-residency metrics.

5.3.5 Efficiency of the Influence Evaluation Strategy

With four *PA*s, four influential parameters per *PA* and four co-residency metrics, a total of 64 r-values were calculated. Since each parameter was tested at ten levels with ten simulation repetitions, therefore, the Pearson's correlation r-values (Section 5.2.2) were calculated with 98 degrees of freedom. The degrees of freedom for each r-value is equal to two less than the number of observations per parameter [14]. The influential parameters' r-values were included in the findings in the previous sections.

According to the categorization of the strength of correlation in Table 5.3, the |r-values| frequency distribution (Figure 5.11) shows that only 25% of the calculated r-values belong to parameters that had relatively weak correlations with the co-residency metrics (|r-value| < 0.40). On the other hand, 75% of the r-values represented stronger correlations (|r-value| ≥ 0.40).

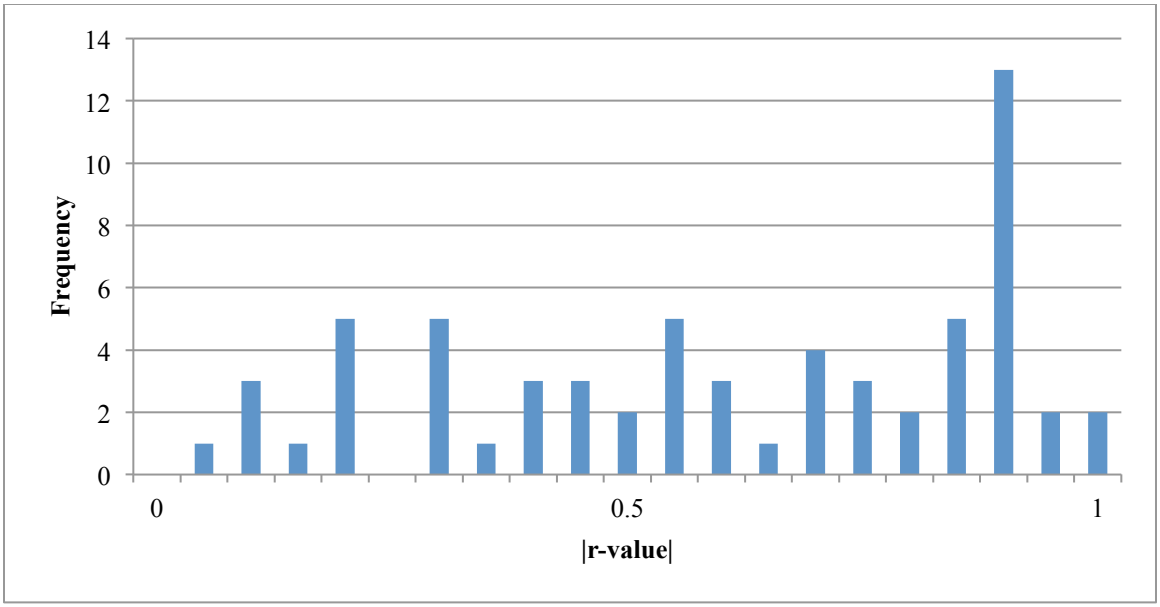


Figure 5.11 A frequency distribution of the influential parameters' |r-values|

5.4 Discussion

The findings in this chapter answered the third research question on how the choice of the influential parameters' settings can reduce the co-residency probability in four *PA*s. However, it is important to emphasize that the following discussion and conclusions are only valid within the range of data collected under the following *PA*s: First Fit, Next Fit, Power Save and Random.

The key results of this chapter are:

- **Co-residency probability can be effectively reduced by the right choice of parameters' settings in the four *PA*s.**

  The results in this chapter show that the settings of the number of hosts and users' arrival rate can positively and negatively affect the co-residency probability. The findings therefore suggest some answers to the third research question depending on the cloud infrastructure size (i.e. the number of hosts) and the cloud population density (i.e. users' arrival rate) in the four *PA*s.

  The following comparison matrix (Table 5.17) summarizes how the co-residency probability has been reduced in different IaaS cloud sizes and population densities in four *PA*s. However, one major observation is that there is no overall best parameter settings or *PA* for reducing co-residency probability.

  This particular finding is relevant particularly to IaaS cloud providers. The finding demonstrates that the impact on co-residency probability should become an important factor in the choice of the parameters' settings and *PA*s for IaaS clouds, an aspect that was not previously present in the available literature.

Reducing the Co-residency Probability	Cloud Size		Cloud Population Density	
	Small (<5000 hosts)	**Large (≥5000 hosts)**	**Low (user arrival rate <3)**	**High (user arrival rate ≥3)**
Reducing the *CCP*	First Fit, Power Save (marginally better)	Next Fit, Random	First Fit, Power Save	
Increasing the *HFL*	Next Fit, Random (preference to Next Fit as number of hosts increase)		Next Fit, Random	First Fit, Power Save
Reducing the *CV*	First Fit, Power Save (consistently at zero)		First Fit, Power Save (consistently at zero)	
Reducing the *CA*	First Fit, Power Save (consistently low at <0.001)	Next Fit, Random (converge to First Fit and Power Save as number of hosts increase	First Fit, Power Save (nearly zero but marginally lower than Next Fit and Random by less than 0.003).	

Table 5.17 The best parameters' settings in four *PA*s to reduce the co-residency probability

- **In general, Next Fit and Random have a better tendency to hinder co-residency in IaaS clouds.**

By comparing the *PA*s at different number of hosts, the probability (with 95% confidence intervals) that a given VM experiences at least one arbitrary co-residency hit is between 0.314 to 0.444 in Next Fit. This is compared to 0.458 to 0.544 in Random, 0.843 to 0.846 in First Fit and 0.838 to 0.841 in Power Save. One possible explanation of why Next Fit and Random are better at reducing the co-residency probability is associated with how they place VMs into hosts. That is, Next Fit and

Random tend to distribute VMs evenly to as many hosts as possible rather than packing them tightly on the first available hosts as First Fit and Power Save do. Therefore, VMs are less vulnerable to the reception of many co-residing VMs in Next Fit and Random, as opposed to First Fit and Power Save. Section 4.6 discusses in detail the similarity between Next Fit and Random in terms of how they place VMs. One can argue that using a *PA* that hinders attackers from achieving co-residency can be a clever action that IaaS cloud provider may take to reduce the avenue for side channel attacks. This finding corroborates the ideas of [79], who recommended that securing against side channel attacks can be achieved via disabling co-residency, that is each VM runs in its dedicated host.

- **For larger IaaS clouds, co-residency probability can be effectively reduced, and even eliminated, by using Next Fit.**

The findings in Section 5.3 showed that as the number of hosts exceeds 25000 hosts, the use of Next Fit as a *PA* eliminated the Co-residency Coverage Probability *CCP* and Co-residency Activity *CA*. The most likely cause of this is that the frequency, at which a given VM *u* receives a co-residency hit when Next Fit is used, is proportional to the number of hosts. That is, once *u* is placed in a host *x*, Next Fit selects *x* for the next placement after placing a VM in all hosts. Therefore, this finding suggests that using Next Fit in larger IaaS clouds (with a larger number of hosts) can be useful for reducing, and even eliminating, the probability of co-residency.

- **VM co-residency probability is dependent on the number of hosts, where IaaS clouds with a higher number of hosts are less likely to exhibit co-residency.**

This finding suggests that VMs hosted in IaaS clouds that have a larger number of hosts are likely to be safer from co-residency compared to IaaS clouds with a smaller number of hosts. Regardless of the *PA*, the disproportionate effect that the number of hosts had on the co-residency metrics (see Figure 5.2, Figure 5.5 and Figure 5.8) provided convincing evidence to support this finding.

One practical application of this finding is for cloud users to compare the potential co-residency probability at different IaaS clouds, depending on the number of hosts in each cloud. On the basis of the previous evidence, it seems fair to suggest that hosting

sensitive data and applications in IaaS clouds that have more hosts can be an effective practice to reduce the probability of experiencing co-residency hits. Such action can reduce the attack surface for side channel attacks. One exception to this recommendation is that the Co-residency Vacancy *CV* was shown to increase as the number of hosts increases under Next Fit and Random *PA*s. However, the existence of a *CV* during a VM's lifetime is considered to be a necessary, but not sufficient, condition for an attacker to achieve co-residency with a given VM (see Section 3.4.3). In addition, IaaS cloud providers might consider adding more hosts as a measure to hinder co-residency. However, this suggests the existence of a trade-off between reducing costs (i.e. not investing in new hosts) and increasing security (i.e. reducing co-residency probability).

- **The Influence Evaluation Strategy was efficient in identifying the most influential parameters on the co-residency metrics.**

With regards to the Influence Evaluation Strategy (Section 4.3.2), one of the limitations of using 2-way Fractional Factorial experimentations is that the effect of each parameter was measured at two levels only. Therefore, there was no guarantee that there will be no outliers between these two levels as the presence of such outliers might impact the strategy's evaluation of the parameters effects. This limitation was addressed in this chapter by examining the most influential parameters at more levels. The results in Section 5.3.5 showed that there were strong linear correlations between the influential parameters and the co-residency metrics. This linear correlation suggests that outliers did not impact the ability of the Influence Evaluation Strategy to identify the most influential parameters.

5.5 Summary

Using the *VMC* simulator as a testbed, the four most influential parameters identified in Chapter 4 were used in controlled experiments in this chapter. These experiments aimed to explore how the most influential parameters' settings in four *PA*s could positively and negatively affect the co-residency metrics. In order to achieve this aim, these experiments estimated the co-residency metrics in four *PA*s under a wide range of likely settings for publicly accessible IaaS clouds (Section 5.2).

Next, Pearson's correlation analysis [14] has been applied to study the correlation between these parameters and the co-residency metrics. This analysis helped in identifying the parameters' settings that were able to reduce the co-residency probability in each *PA* (see Table 5.17).

Based on this finding, Section 5.4 presents evidence that VMs hosted in IaaS clouds with a higher number of hosts are less likely to exhibit co-residency. Further, using Next Fit in larger IaaS clouds has been shown to reduce effectively, and even eliminate, the co-residency probability. In addition, the four *PA*s have been compared in their ability to reduce the co-residency probability. For instance, VMs in IaaS clouds that use Next Fit or Random are found to be more resilient against receiving co-resident VMs compared to when First Fit or Power Save are used. By comparing the *PA*s at different number of hosts, the probability (with 95% confidence intervals) that a given VM experiences at least one arbitrary co-residency hit is between 0.314 to 0.444 in Next Fit. This is compared to 0.458 to 0.544 in Random, 0.843 to 0.846 in First Fit and 0.838 to 0.841 in Power Save.

This chapter focused on reducing the co-residency probability (i.e. the chance that a VM experiences an arbitrary co-residency hit). In contrast, Chapter 6 estimates the malicious co-residency probability (i.e. the chance that a VM experiences a malicious co-residency hit).

Chapter 6
Analytical Estimation of Malicious Co-residency Probability

6.1 Introduction

Chapter 3 defines four co-residency metrics that estimate probabilities related to co-residency hits from arbitrary VMs (i.e. hits from malicious and honest VMs). Chapter 4 quantified the influence of cloud parameters on the co-residency metrics then identifies the most influential parameters and 2-parameter interactions (i.e. the second research question). Next, Chapter 5 answered the third research question and provides simulation estimates of the co-residency metrics. These estimates helped to find the best parameter settings that reduce the probability of receiving honest and malicious co-resident VMs in four *PA*s. On the other hand, this chapter is concerned with estimating the probability that a new co-residing VM belongs to an attacker (i.e. malicious co-residency probability) with the coexistence of different populations of attackers.

Two malicious co-residency metrics are defined to estimate probabilities related to malicious co-residency (i.e. the fourth research question). These probabilities are the probability that a VM u receives a malicious hit and for how long it remains free from malicious hits. The malicious co-residency metrics take into account the "biggest unknown" in the attack model: the ratio of attackers VMs, noted to as α, which can take any value between 0 and 1 (see Section 3.2). Unlike the co-residency metrics, this very wide range of possible values of α presents a challenge in using the *VMC* simulator to estimate the malicious co-residency metrics (see Section 1.5). Instead, this Chapter provides approximate analytical estimates of the malicious co-residency metrics that take α into account. These estimates are derived to explore all likely values of α easily, an attempt that simulation and time limitations did not allow. These estimates are then used to determine analytically, with the coexistence of α attacking VMs, the best *PA*s that reduce the probability that a new co-residing VM belongs to an attacker.

To validate these analytical approximations, the *VMC* simulator is used to estimate the malicious co-residency metrics under a specific α value using First Fit, Next Fit, Power Save and Random.

The remainder of this chapter is organized as follows. The next section derives approximate analytical estimates of two malicious co-residency metrics that take α into account. Section 6.3 defines how the proposed analytical approximation is validated using the *VMC* simulator. Further, Section 6.4 describes the experiment's settings. Finally, the findings are presented in Section 6.4 and discussed in Section 6.5.

6.2 Malicious Co-residency Metrics

As pointed out in Chapter 1, the risk of side channel attacks is magnified by the occurrence of malicious co-residency hits only. The second hypothesis of this thesis states "for a given VM, there is a non-zero probability that a new co-residing VM belongs to an attacker for any of the four *PA*s considered." While the previous co-residency metrics (i.e. *CCP, HFL, CV* and *CA*) address co-residency hits caused by arbitrary VMs, they do not distinguish between malicious hits originated by attackers and honest hits. Therefore, this section defines two malicious co-residency metrics to estimate probabilities related to malicious co-residency hits. These probabilities are (1) the probability that a VM *u* receives a malicious hit and (2) for how much time it remains free from malicious hits.

Based on the system and attack models (see Section 3.2), this section derives approximate analytical estimates of the malicious co-residency metrics that take α into account.

It will be necessary to note that before proceeding to define the malicious co-residency metrics, analytical approximations in this chapter are mainly based on the Probability theory. In particular, *P(x)* is used as a notation for probability (i.e. *P(x)* reads as the probability of x). In addition, the symbol "|" is used in conjunction with *P(x)* to denote a conditional probability (e.g. *P*(*x*|*y*) means "probability of *x* given condition *y*"). Further, the notation *P(A and B)* is interpreted as *P(A) * P(B|A)*. In addition, the symbol $\hat{E}(x)$ is used to refer to an estimated value of variable *x*.

6.2.1 Preliminary Definitions

In addition to the definitions set forth in Chapter 3, the following are important definitions related to the malicious co-residency metrics estimations.

Expected Number of Hits $\widehat{\boldsymbol{E}}(\boldsymbol{k}|\boldsymbol{k}>0)$

For a given VM *u* that receives at least one hit ($k>0$), the $\hat{E}(k|k>0)$ calculates the total number of honest and malicious hits that *u* experiences. In addition, $P(k=K\,|k>0)$ is the

probability that *u* receives *K* hits, given that it receives at least one hit. Further, n_K is the number of VMs that experienced *K* hits and n_{hit} is the total number of VMs that experienced at least one hit. Accordingly, the expected number of hits $\hat{E}(k|k>0)$ can be derived as follows:

$$\widehat{\boldsymbol{E}}(\boldsymbol{k}|\boldsymbol{k}>\boldsymbol{0}) = \sum_{\boldsymbol{K}=\boldsymbol{1}}^{\infty} \boldsymbol{K} * \boldsymbol{P}(\boldsymbol{k}=\boldsymbol{K}\,|\boldsymbol{K}>\boldsymbol{0})$$

$$\widehat{\boldsymbol{E}}(\boldsymbol{k}|\boldsymbol{k}>\boldsymbol{0}) = \sum_{\boldsymbol{K}=\boldsymbol{1}}^{\infty} \boldsymbol{K} * \frac{\boldsymbol{n_K}}{\boldsymbol{n_{hit}}}$$

$$\widehat{\boldsymbol{E}}(\boldsymbol{k}|\boldsymbol{k}>\boldsymbol{0}) = \frac{\boldsymbol{1}}{\boldsymbol{n_{hit}}} \sum_{\boldsymbol{K}=\boldsymbol{1}}^{\infty} \boldsymbol{K} * \boldsymbol{n_K} \quad , \widehat{\boldsymbol{E}}(\boldsymbol{k}|\boldsymbol{k}>\boldsymbol{0}) \geq \boldsymbol{1}$$

Expected K^{th} Lifetime Ratio $\widehat{E}(L_k)$

For a given VM *u,* the expected K^{th} lifetime ratio L_K can be derived using the expected number of hits $\hat{E}(k|k>0)$ as follows (n_K as the number of VMs that experienced *K* hits):

$$\widehat{\boldsymbol{E}}(\,\boldsymbol{L_K}\,) = \frac{\boldsymbol{1}}{\boldsymbol{n_K}} \sum_{\boldsymbol{u}=\boldsymbol{1}}^{\boldsymbol{n_K}} \boldsymbol{L_{K_u}} \qquad , \quad 1 \leq \mathrm{K} \leq \widehat{\boldsymbol{E}}(\boldsymbol{k}|\boldsymbol{k}>\boldsymbol{0})$$

The above can be used to calculate the expected lifetime ratios up to the $\hat{E}(k|k>0)^{th}$ hit (Figure 3.2). The $L_{\hat{E}(k|k>0)+1}$ lifetime ratio, which represents the portion of the lifetime between the $\hat{E}(k|k>0)^{th}$ hit and the time at which a VM ends, can be derived as follows:

$$\widehat{\boldsymbol{E}}(\,\boldsymbol{L}_{\,\widehat{\boldsymbol{E}}(\boldsymbol{k}|\boldsymbol{k}>\boldsymbol{0})+\boldsymbol{1}}\,) = \boldsymbol{1} - \sum_{\boldsymbol{K}=\boldsymbol{1}}^{\widehat{\boldsymbol{E}}(\boldsymbol{k}|\boldsymbol{k}>\boldsymbol{0})} \widehat{\boldsymbol{E}}(\,\boldsymbol{L_K}\,)$$

6.2.2 Analytical Estimation of the Malicious Co-residency Metrics

Having defined α, $\hat{E}(k|k>0)$ and $\hat{E}(L_K)$, the next sections define the malicious co-residency metrics and show how their estimates are derived using an analytical approximation.

6.2.2.1 Malicious Co-residency Probability (*MCP*)

The Malicious Co-residency Probability *MCP* is the probability that an honest VM *u* encounters a malicious co-residency hit at least once during its lifetime. The estimation of *MCP* extends the Co-residency Coverage Probability *CCP* metric (see Section 3.4.1). Unlike the *CCP,* the *MCP* focuses on malicious co-residency hits (caused by malicious VMs only), whereas, the *CCP* considers both malicious and honest co-residency hits.
With the use of the *CCP*, α and $\hat{E}(k|k>0)$, Probability theory is applied to derive an estimate approximation of the *MCP* for a given VM *u*:

∀ CCP, α and $\hat{E}(k|k>0)$***: 0 ≤ CCP ≤ 1, 0 ≤ α ≤ 1 and*** $\hat{E}(k|k>0) \geq 1$***:***

$\hat{E}(MCP)$ *= P(at least 1 hit AND at least 1 hit is malicious | number of hits>0)*

*= P (k >0) * P(at least 1 hit is malicious | k >0)*

*= CCP * (1 – P (all hits are honest | k >0))*

At this point, an approximation is introduced to the MCP estimate that involves assuming that every VM, that is hit at least once, is hit exactly $\hat{E}(k|k>0)$ times (where $\hat{E}(k|k>0)$ will be rounded down to the nearest integer):

$\approx$ *CCP * (1 – P (all* $\hat{E}(k|k>0)$ *hits are honest))*

$\approx$ *CCP * (1 – (P (*hit_1 *is honest) * . . . * P (*$hit_{\hat{E}(k|k>0)-1}$ *is honest) *

P ($hit_{\hat{E}(k|k>0)}$ *is honest)))*

$\approx$ *CCP * (1-P (hit is non-malicious)* $^{\hat{E}(k|k>0)}$*)*

$\approx$ *CCP * (1- (1-α)* $^{\hat{E}(k|k>0)}$*)*

Therefore, the Malicious Co-residency Probability *MCP* estimate can be approximated as follows:

$$\widehat{E}(MCP) \approx CCP * (1- (1-\alpha)^{\widehat{E}(k|k>0)}) \quad , 0 \leq \widehat{E}(MCP) \leq 1$$

The *MCP* can reach its maximum value when every VM in the cloud will certainly has all of its *k* co-residency hits as malicious. In this particular case, the *MCP* becomes the same as the *CCP* value. This scenario can manifest itself when all other VMs in the cloud are malicious VMs (i.e. the attacker's VM requests ratio α equals one). However, this requires an attacker to originate and control all the VMs in the cloud in order to achieve co-residency hits with target VMs. In contrast, the *MCP* can reach its minimum value (i.e. zero) when every VM in the cloud will certainly have all of its *k* co-residency hits as honest. This can be the case when the attacker's VM requests ratio α is zero. Another scenario in which the *MCP* can reach zero is when each VM ends up running solely in its own physical host (i.e. in this case *CCP* equals zero). This scenario is suggested in [79] to disable the risk of side channel attacks. However, this requires the customer to pay for the opportunity cost of under-utilizing the hosts' resources due to not sharing them with other cloud users (see Section 2.3.3.1).

6.2.2.2 Attacker-free Lifetime Ratio (*AFL*)

For a given VM *u*, the Attacker-free Lifetime Ratio *AFL* is the sum of the lifetime ratios (Figure 3.2) where *u* is free of malicious co-residency hits. A lifetime ratio L_K is considered to be attacker-free when the *K-1*th hit and all previous hits are honest.

Unlike the Hit-free Lifetime Ratio *HFL* metric (see Section 3.4.2), the *AFL* calculates the attacker-free lifetime ratio from the moment VM *u* is launched until it experiences the first malicious hit. On the other hand, the *HFL* calculates the lifetime ratio from the moment *u* is launched until it experiences the first hit; regardless of whether the first hit is malicious or honest. The AFL_u for a VM *u* that experiences at least one co-residency hit (*k>0)* can be estimated using simulation, for example, as follows:

$$AFL_u = \sum_{K=1}^{k+1} (L_K | L_K \text{ is attacker free}) \quad , 0 \leq AFL_u \leq 1$$

The following is an approximate analytical estimate of the *AFL* that extends the *CCP* metric:

$\forall$ *CCP,* α, $\hat{E}(k|k>0)$*and* $\hat{E}(L_K)$*:* $0 \leq CCP \leq 1$, $0 \leq \alpha \leq 1$, $\hat{E}(k|k>0) \geq 1$ *and* $0 \leq \hat{E}(L_K) \leq 1$*:*

$\widehat{\boldsymbol{E}}(\boldsymbol{AFL})$= *(AFL | no hit) + (AFL | at least 1 hit and all hits are honest)*

= (AFL when u receives no hit) P(no hit) + (AFL| k>0 and all hits are honest)*

*= 1 * (1-CCP) + (AFL | k>0 and all hits are honest)*

*= (1-CCP) + P(k>0) * (AFL | all are honest hits)*

*= (1-CCP) + CCP * (AFL | all are honest hits)*

At this point, approximations are introduced that involve assuming that every VM *u* that is hit at least once is hit exactly $\hat{E}(k|k>0)$ times where $\hat{E}(k|k>0)$ will always be rounded down to the nearest integer. As a result, *u* has a total of $\hat{E}(k|k>0)$+1 lifetime ratios $\hat{E}(L_1), \ldots, \hat{E}\left(L_{\hat{E}(k|k>0)}\right), \hat{E}(L_{\hat{E}(k|k>0)+1})$:

≈ *(1-CCP) + CCP * (AFL | all* $\hat{E}(k|k>0)$ *hits are honest)*

≈ *(1-CCP) + CCP * (*$\hat{E}(L_1)$ + $\hat{E}(L_2|hit_1$ *non-malicious)* + $\hat{E}(L_3|hits_{1\ and\ 2}$ *non-malicious)* +

… + $\hat{E}$ *(*$L_{\hat{E}(k|k>0)+1}|hits_{\ 1\ to\ \hat{E}(k|k>0)}$ *non-malicious))*

≈ *(1-CCP) + CCP * (*$\hat{E}(L_1)$ + $\hat{E}(L_2)$ ** P(*hit_1 *non-malicious)* +

$\hat{E}(L_3)$ ** P(*hit_1 *non-malicious)* P(*hit_2 *non-malicious)* + … +

$\hat{E}\left(L_{\hat{E}(k|k>0)+1}\right)$ ** P(*$hits_{\ 1\ to\ \hat{E}(k|k>0)}$ *non-malicious))*

≈ *(1-CCP) + CCP * (*$\hat{E}(L_1) + \hat{E}(L_2) * (1-\alpha) + \hat{E}(L_3) * (1-\alpha)^2 + \ldots +$

$\hat{E}\left(L_{\hat{E}(k|k>0)}\right) * (1-\alpha)^{\hat{E}(k|k>0)-1} + \hat{E}\left(L_{\hat{E}(k|k>0)+1}\right) * (1-\alpha)^{\hat{E}(k|k>0)}$)

Therefore, the *AFL* estimate can be approximated as follows:

$$\widehat{\boldsymbol{E}}(\boldsymbol{AFL}) \approx (\boldsymbol{1-CCP}) + \boldsymbol{CCP} * \sum_{\boldsymbol{K=1}}^{\widehat{\boldsymbol{E}}(\boldsymbol{k}|\boldsymbol{k}>\boldsymbol{0})+\boldsymbol{1}} \left[\widehat{\boldsymbol{E}}(\boldsymbol{L_K}) * (\boldsymbol{1-\alpha})^{\boldsymbol{K-1}}\right] \quad , \boldsymbol{0} \leq \widehat{\boldsymbol{E}}(\boldsymbol{AFL}) \leq \boldsymbol{1}$$

The *AFL* can reach its maximum value when there are no attackers in the IaaS cloud (i.e. the attackers VM requests ratio α is zero). This action can result in a situation where every VM in the cloud will have all of its *k* hits as honest. In contrast, the *AFL* can reach its minimum value (i.e. *HFL* or zero) when every new VM request in the cloud is malicious (i.e. the

attackers VM requests ratio α is one). As mentioned in the previous section, this requires the attacker to originate and control all the VMs in the cloud in order to achieve malicious co-hits with target VMs. Another possible scenario where the *AFL* can reach its minimum value is possible. For instance, when every sequence of newly created VMs tends to be placed in the same physical host until the host becomes full (i.e. no space for new VMs). This scenario is shown by [79] to be very dependent on the *PA* that is used by the cloud provider.

6.3 Method

Section 6.2 derived approximate analytical estimation of the malicious co-residency metrics in order to easily examine all likely values of α values (the ratios of attackers VMs requests). A comparison was made with experimental estimates obtained using simulation to validate these analytical estimates. The *VMC* simulator was used to estimate the *MCP* and *AFL* with α set to 0.10 in a variety of IaaS clouds settings (i.e. different Number of Hosts (X2) and Users' Arrival Rates (X4)). These simulation estimates can help to determine how good the malicious co-residency metrics analytical approximations are.

Based on the description in Sections 6.2.2.1 and 6.2.2.2, the analytical estimates of *MCP* and *AFL* were calculated using the simulation estimates (Table 6.1 and Table 6.2): (1) the Co-residency Coverage Percentage $\hat{E}(CCP)$, (2) number of hits $\hat{E}(k|k>0)$ and (3) life ratios $\hat{E}(L_k)$. These estimates were calculated during the simulation experiments in this chapter:

Placement Algorithms	$\widehat{\boldsymbol{E}}(\boldsymbol{CCP})$	$\widehat{\boldsymbol{E}}(k\|k>0)$	$\widehat{\boldsymbol{E}}(\boldsymbol{L_1})$	$\widehat{\boldsymbol{E}}(\boldsymbol{L_2})$	$\widehat{\boldsymbol{E}}(\boldsymbol{L_3})$	$\widehat{\boldsymbol{E}}(\boldsymbol{L_4})$	$\widehat{\boldsymbol{E}}(\boldsymbol{L_5})$
First Fit	0.851	4.796	0.113	0.241	0.137	0.109	0.400
Next Fit	0.394	3.033	0.729	0.103	0.034	0.134	--
Power Save	0.851	4.705	0.110	0.230	0.132	0.108	0.420
Random	0.537	3.090	0.363	0.348	0.098	0.191	---

Table 6.1 Important estimates obtained by the *VMC* simulator with Number of Hosts (X2) varying between 1000-30000

Placement Algorithms	$\widehat{E}(CCP)$	$\widehat{E}(k\|k>0)$	$\widehat{E}(L_1)$	$\widehat{E}(L_2)$	$\widehat{E}(L_3)$	$\widehat{E}(L_4)$	$\widehat{E}(L_5)$
First Fit	0.815	4.098	0.127	0.293	0.126	0.093	0.361
Next Fit	0.848	3.363	0.327	0.257	0.153	0.263	---
Power Save	0.811	4.103	0.132	0.288	0.125	0.095	0.360
Random	0.842	3.753	0.262	0.250	0.156	0.331	---

Table 6.2 Important estimates obtained by the *VMC* simulator with Users' Arrival Rate (X4) varying between 2-5

On the other hand, the *VMC* simulator estimates the malicious co-residency metrics for a certain α as follows:

- **Malicious Co-residency Probability *MCP*:**

  Let n be the total number of created VMs in the cloud and $n_{hit\ by\ malicious}$ is the total number of VMs that experienced at least one malicious hit, then the *MCP* was estimated using simulation as follows:

$$\boldsymbol{MCP = \frac{n_{hit\ by\ malicious}}{n}}$$

- **Average Attacker-free Lifetime Ratio *AFL*:**

  Let n_{hit} be the total number of VMs that experienced at least one hit (k>0), then the *AFL* was estimated using simulation as follows:

$$\boldsymbol{AFL = \frac{1}{n}\left[\sum_{u=1}^{n_{hit}} AFL_u + (n - n_{hit})\right]}$$

With α of 0.10, a total of 80 simulation estimates of the *MCP* and *AFL* were obtained under different numbers of hosts and the users' arrival rates (Table 6.3), while keeping the remaining parameters constant (Table 5.2). Then, the average of these estimates was compared with the metrics' analytical prediction under First Fit, Next Fit, Power Save and Random in Section 6.4.1.

Number of Hosts (X2)	Users' Arrival Rates (X4)
1000	2
10000	3
15000	4
30000	5

Table 6.3 The parameters levels used in the *VMC* simulator to estimate the *MCP* and *AFL*

6.3.1 Analytical Estimation Accuracy

In approximation theory [80], the predicted values can often be overestimation or underestimation of the actual measurements. This can result from the fact that an approximation cannot include all the parameters that represent the predicted reality. Thus, this thesis calculates the percentage difference to quantify this amount of error between the approximate analytical estimates and the simulation estimates. The percentage difference can be obtained as follows:

$$\textbf{Percentage difference} \approx \frac{\textbf{analytical estimate} - \textbf{simulation estimate}}{\textbf{simulation estimate}} * \mathbf{100}$$

Approximation theory states that an accepted analytical estimate depends on the type of application and the sensitivity of the predicted values [80]. For the convenience of this experiment, an analytical estimate will be considered acceptable if it has a percentage difference up to ±15%.

6.4 Findings

This section outlines the findings concerning the analytical estimation validation as well as an analysis of malicious co-residency probabilities under different attackers ratios α.

6.4.1 Analytical Estimation Validation

With an attacker ratio α of 0.10, and under different number of hosts and users' arrival rates, the following (Table 6.4, Table 6.5, Table 6.6 and Table 6.7) show the analytical estimates, the simulation estimates and the corresponding percentage difference for both the *MCP* and *AFL* in four *PA*s.

Placement Algorithms	$\widehat{E}(MCP)$ Analytical estimate	$\widehat{E}(MCP)$ Simulation estimate	Percentage Difference
First Fit	0.3377	0.2554	32.22%
Next Fit	0.1077	0.1056	2.00%
Power Save	0.3328	0.2774	19.95%
Random	0.1492	0.1679	-11.13%

Table 6.4 Percentage differences of the *MCP* estimates with an α of 0.10 as Number of Hosts (X2) varies between 1000-30000

Placement Algorithms	$\widehat{E}(MCP)$ Analytical estimate	$\widehat{E}(MCP)$ Simulation estimate	Percentage Difference
First Fit	0.2857	0.2846	0.39%
Next Fit	0.2528	0.2828	-10.58%
Power Save	0.2845	0.2877	-1.12%
Random	0.2749	0.2895	-5.06%

Table 6.5 Percentage differences of the *MCP* estimates with an α of 0.10 as Users' Arrival Rate (X4) varies between 2-5

Placement Algorithms	$\widehat{E}(AFL)$ Analytical estimate	$\widehat{E}(AFL)$ Simulation estimate	Percentage Difference
First Fit	0.797	0.826	-3.53%
Next Fit	0.979	0.945	3.51%
Power Save	0.795	0.824	-3.61%
Random	0.943	0.929	1.46%

Table 6.6 Percentage differences of the *AFL* estimates with an α of 0.10 as Number of Hosts (X2) varies between 1000-30000

Placement Algorithms	$\widehat{E}(AFL)$ Analytical estimate	$\widehat{E}(AFL)$ Simulation estimate	Percentage Difference
First Fit	0.833	0.842	-1.10%
Next Fit	0.887	0.902	-1.74%
Power Save	0.834	0.841	-0.91%
Random	0.863	0.885	-2.58%

Table 6.7 Percentage differences of the *AFL* estimates with an α of 0.10 as Users' Arrival Rate (X4) varies between 2-5

The previous tables showed an agreement between the analytical estimates and the simulation estimates across all *PA*s with an α of 0.10. About 75% and 100% of the obtained analytical estimates of the *MCP* and *AFL,* respectively, had percentage differences less than 15%. Moreover, the mean percentage differences are 10.31% and 2.31% for the *MCP* and *AFL*, respectively. On the other hand, the *MCP* was overestimated in First Fit and Power Save as shown in the percentage difference that increased to levels that were pre-defined as not being adequate (Section 6.3.1). Appendix E outlines, in detail, the *VMC* simulator's

estimates of the malicious co-residency metrics under different numbers of hosts and users' arrival rates with α set to 0.10 in four *PA*s.

6.4.2 Malicious Co-residency Metrics as Attackers Ratio α Varies

This section presents the approximate analytical estimation of the *MCP* and *AFL* under different α values, where these analytical estimates were calculated using the simulation estimates from Table 6.1 and Table 6.2.

Table 6.8 shows the analytical estimates of the *MCP* under different α values.

As Figure 6.1 illustrates, the relationship between the *MCP* and the attackers' VM requests ratio α depends on the used *PA*. When 0.1 or less α is present in the IaaS cloud, the expected *MCP* for First Fit, Next Fit, Power Save and Random are very close. However, when α is greater than 0.1, Next Fit and Random significantly outperform (in reducing the *MCP*) the rest of the *PA*s.

In addition, Next Fit outperforms (in reducing the *MCP*) the rest of the *PA*s for all amounts of α. Surprisingly, even with attackers' VM requests ratio of 0.99, Next Fit was able to limit the probability of malicious co-residency to only 0.394.

	***MCP* estimate using analytical prediction**			
α	**First Fit**	**Next Fit**	**Power Save**	**Random**
0	0	0	0	0
0.000000001	4.081×10^{-9}	1.195×10^{-9}	4.004×10^{-9}	1.659×10^{-9}
0.0000001	4.081×10^{-7}	1.195×10^{-7}	4.004×10^{-7}	1.659×10^{-7}
0.000001	4.081×10^{-6}	1.195×10^{-6}	4.004×10^{-6}	1.659×10^{-6}
0.00001	4.081×10^{-5}	1.195×10^{-5}	4.004×10^{-5}	1.659×10^{-5}
0.0001	0.000408	0.000119	0.000400	0.000166
0.001	0.0041	0.00119	0.00400	0.00166
0.01	0.0400	0.01183	0.03930	0.01642
0.1	0.3376	0.10777	0.33263	0.14922
0.2	0.5592	0.19375	0.55317	0.26752
0.3	0.6972	0.26044	0.69210	0.35863
0.4	0.7776	0.31032	0.77406	0.42622
0.5	0.8204	0.34586	0.81837	0.47393
0.6	0.8405	0.36954	0.83958	0.50535
0.7	0.8484	0.38378	0.84805	0.52399
0.8	0.8506	0.39101	0.85056	0.53328
0.9	0.8510	0.39363	0.85098	0.53656
0.99	0.8510	0.39400	0.85100	0.53700

Table 6.8 *MCP* estimates using analytical prediction as α varies

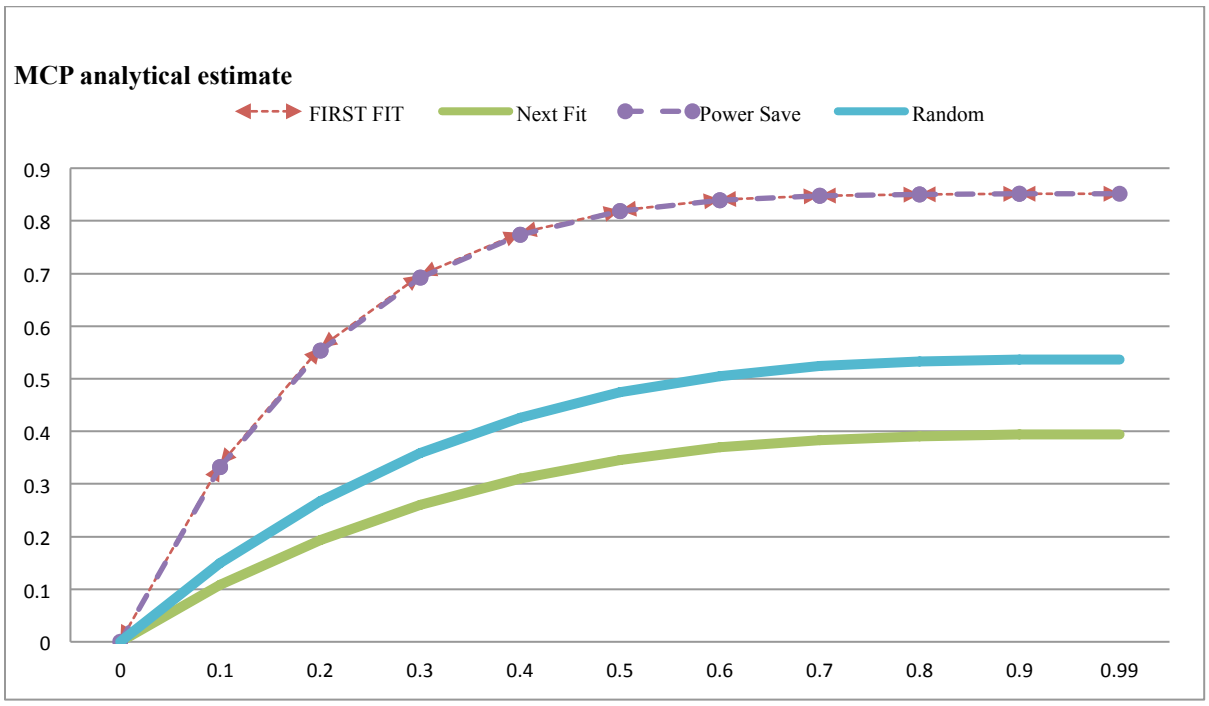


Figure 6.1 Variation of *MCP* with attackers' VM requests ratio α

Moreover, Table 6.9 shows the Analytical estimates of the *AFL* (the expected Attacker-Free Lifetime ratio of a given VM) across different α values.

Similar to the *MCP*, Figure 6.2 illustrates that the relationship between the *AFL* and the attackers' VM requests ratio α depends on the used *PA*. When 0.1 or less α is available, the expected *AFL* for First Fit, Next Fit, Power Save and Random are very close. However, when α is greater than 0.1, Next Fit and Random significantly outperform (in increasing the *AFL*) the rest of the *PA*s.

In addition, Next Fit outperforms (in increasing the *AFL*) the rest of the *PA*s for all amounts of α. Even with attackers' VM requests ratio of 0.99, Next Fit was able to prolong the lifetime ratio at which a given VM is safe from malicious hits to 0.894.

	***AFL* using analytical prediction**			
α	**First Fit**	**Next Fit**	**Power Save**	**Random**
0	1.000	1.000	1.000	1.000
0.000000001	1.000	1.000	1.000	1.000
0.0000001	1.000	1.000	1.000	1.000
0.000001	1.000	1.000	1.000	1.000
0.00001	1.000	1.000	1.000	1.000
0.0001	1.000	1.000	1.000	1.000
0.001	0.998	1.000	0.998	0.999
0.01	0.977	0.998	0.977	0.994
0.1	0.797	0.979	0.794	0.943
0.2	0.648	0.961	0.643	0.893
0.3	0.539	0.946	0.533	0.849
0.4	0.460	0.934	0.453	0.810
0.5	0.401	0.923	0.394	0.777
0.6	0.356	0.915	0.350	0.747
0.7	0.321	0.908	0.315	0.721
0.8	0.292	0.902	0.287	0.698
0.9	0.267	0.897	0.263	0.677
0.99	0.247	0.894	0.245	0.660

Table 6.9 *AFL* estimates using analytical prediction as α varies

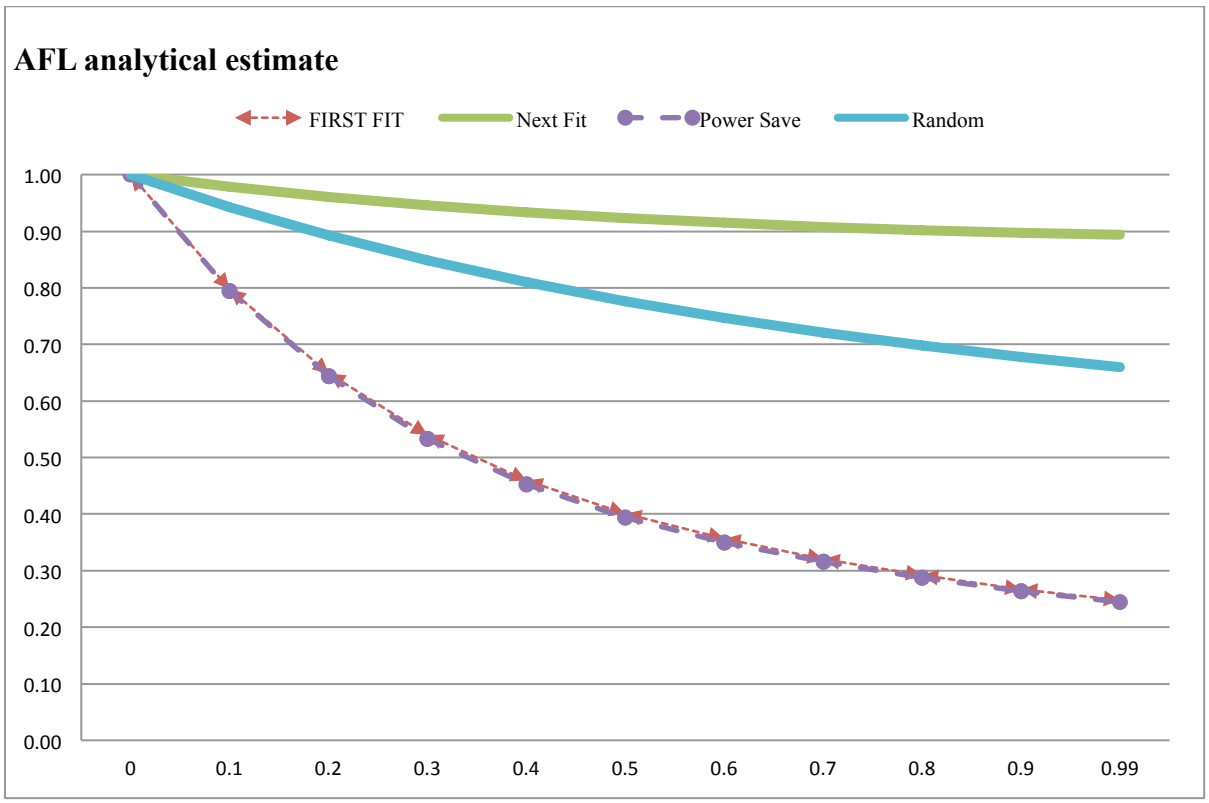


Figure 6.2 Variation of *AFL* with attackers' VM requests ratio α

6.5 Discussion

Given that attackers are present in a given IaaS cloud (i.e. VMs requests ratio α is greater than zero), the findings in this chapter (Table 6.8 and Table 6.9) validated the second research hypothesis. This hypothesis states "for a given VM, there is a non-zero probability that a new co-residing VM belongs to an attacker for any of the four *PA*s considered." The findings also provided useful insights to answer the fourth research question that states "for a given VM, what is the probability that a new co-residing VM belongs to an attacker." Further, the results illustrate how the malicious co-residency probability varies in various populations of attacker VMs.

It is important to emphasize that the following discussion and conclusions are only valid within the range of data collected under the following *PA*s: First Fit, Next Fit, Power Save and Random.

The key findings in this chapter are:

- **The approximate analytical estimates of the malicious co-residency metrics are acceptable over any given proportion of malicious users.**

  The results in (Table 6.4, Table 6.5, Table 6.6 and Table 6.7) show the analytical estimates, the simulation estimates and the corresponding percentage differences. About 75% and 100% of the obtained analytical estimates of the *MCP* and *AFL,* respectively*,* had percentage differences less than 15% in the four *PA*s. Moreover, the mean percentage differences are 10.31% and 2.31% for the *MCP* and *AFL*, respectively. On the other hand, the *MCP* was overestimated in First Fit and Power Save as shown in the percentage difference that increased to levels that were pre-defined as not being adequate (Section 6.3). As these results show, the analytical estimation of the malicious co-residency metrics agreed with the experimental estimate (i.e. using the *VMC* simulator). Thus, this finding allows the conclusion that the analytical estimation derived in this chapter can become very useful for estimating the probability of an attacker successfully co-residing with a given VM under any α value.

  There can be several useful applications of the proposed analytical estimation. For IaaS cloud providers, the *VMC* simulator can be used to find the *CCP* metric and the $\hat{E}(k|k > 0)$ (Section 6.2) in order to obtain an analytical estimate of the malicious co-residency metrics under any α value. This action can reveal valuable insights into the IaaS cloud under study and can be used to compare the malicious co-residency occurrence probabilities in different cloud settings, *PA*s and different attacker ratios.

- **Under different proportion of malicious users, the right choice of *PA* can hinder attackers from easily achieving malicious co-residency.**

The findings in Section 6.4.2 suggest that the used *PA* is a primary factor in determining the malicious co-residency likelihood. However, the results showed that there is no best *PA* for reducing the malicious co-residency probability. For instance, Next fit and Random were better in reducing this probability compared to First Fit and Power Save. By comparing the *PA*s as α varies between 0 to 0.99 (Table 6.8 and Table 6.9), the probability (with 95% confidence intervals) that an honest VM u encounters a malicious co-residency hit at least once during its lifetime (i.e. the *MCP*) is between 0.197 to 0.376 in Next Fit, compared to 0.270 to 0.514 in Random, 0.490 to 0.862 in First Fit and 0.487 to 0.860 in Power Save. One possible explanation of

why Next Fit and Random are better at reducing the co-residency probability was presented in Section 5.4. The explanation suggests that Next Fit and Random tend to distribute VMs evenly across as many hosts as possible rather than packing them tightly on the first available hosts as First Fit and Power Save do. Therefore, VMs are less vulnerable to the reception of many co-residing VMs in Next Fit and Random, as opposed to First Fit and Power Save. Thus, this provides evidence that the right choice of *PA*s can hinder attackers from easily achieving malicious co-residency. In addition, this finding corresponds well with the conclusion made in Section 5.4 that the impact on co-residency probability should become an important factor in the choice of *PA*s for IaaS clouds.

- **Generating 40% of the VMs requests –by attackers- in a given IaaS cloud can lead to a substantial increase in the chance of achieving malicious co-residency.** The findings in Section 6.4.2 suggest that attackers can effectively increase their chance to achieve malicious co-residency by originating no more than 40% of the VMs requests in a given IaaS cloud. The results in (Table 6.8) show that increasing the attacker's VMs request ratio from 0 to 0.4 caused a significant increase in the *MCP*. For instance, the *MCP* increased from 0 to reach about 0.77 in First Fit and Power Save, 0.31 in Next Fit and 0.42 in Random. In addition, the analytical estimation of the *AFL* provides valuable insights that can help the attacker to increase the chance of achieving co-residency with a particular VM *u*. For example, the attacker can time the VM requests during a particular duration of *u*'s lifetime during which *u* is expected not to be hit-free.
  On the basis of the evidence currently available, it seems fair to suggest that organized attackers with plentiful resources (e.g. organization-sponsored attackers) can increase their chance of co-residing with victim VMs. This can be achieved simply by requesting as many VMs as possible. Therefore, it can be argued that the first line of defence against malicious co-residency in IaaS clouds is cloud providers themselves. This action supports the conclusions from Chapter 4 and 5 that IaaS cloud providers must consider selecting a *PA* that hinders attackers from achieving malicious co-residency. In addition, IaaS cloud providers can use the proposed analytical estimation to experiment with a different α in order to determine the range of α ratios that is relatively acceptable to keep the malicious co-residency probability

at its minimum. The providers can set the maximum number of VMs that a user can create using this knowledge. Amazon EC2 limits the number of concurrent VMs a user can create in a single individual account to 20 VMs [5].

6.6 Summary

The risk of side channel attacks is magnified enormously if an honest VM is co-resided by an attacker VM. Therefore, this chapter investigated estimating the probability that the next co-residing VM belongs to an attacker (i.e. the malicious co-residency probability). This estimation was an attempt to answer the fourth research question (i.e. for a given VM, what is the probability that a new co-residing VM belongs to an attacker). This chapter defined two metrics (i.e. the *MCP* and *AFL*) that describe probabilities related to malicious co-residency and take into account the attackers VMs requests ratio α. This thesis is the first to derive two analytical estimates of probabilities related to malicious co-residency in Section 6.2.

Then, analytical estimates of the *MCP* and *AFL* have been compared with experimental estimates (i.e. using the *VMC* simulator) in four *PA*s under an α value of 0.10. The results in (Table 6.4, Table 6.5, Table 6.6 and Table 6.7) show the analytical estimates, the simulation estimates and the corresponding percentage differences in four *PA*s. About 75% and 100% of the obtained analytical estimates of the *MCP* and *AFL,* respectively*,* had percentage differences less than 15% in the four *PA*s. Moreover, the mean percentage differences are 10.31% and 2.31% for the *MCP* and *AFL*, respectively. On the other hand, the *MCP* was overestimated in First Fit and Power Save as shown in the percentage difference that increased to levels that were pre-defined as not being adequate (Section 6.3). Therefore, the derived analytical estimates were shown to agree with the experimental estimates in the four *PA*s in Section 6.4.1.

Further, Section 6.4.2 used the calculated analytical estimates to compare First Fit, Next Fit, Power Save and Random over a wide range of α values. By comparing the *PA*s as α varies between 0 to 0.99 (Table 6.8 and Table 6.9), the probability (with 95% confidence intervals) that an honest VM u encounters a malicious co-residency hit at least once during its lifetime (i.e. the *MCP*) is between 0.197 to 0.376 in Next Fit, compared to 0.270 to 0.514 in Random, 0.490 to 0.862 in First Fit and 0.487 to 0.860 in Power Save. These results seem to be in favour of the second research hypothesis that states “there is a non-zero probability that a new co-residing VM belongs to an attacker in all *PA*s.”

Thus, the aforementioned findings demonstrated that VMs hosted in IaaS clouds that use Next Fit or Random are less likely to receive co-resident attacker VMs compared to when First Fit or Power Save are used. The findings also suggest that the right choice of *PA*s can reduce the probability of being co-resided by attackers VMs, which can reduce the attack surface for side channel attacks. However, an interesting finding in Section 6.4.2 shows that a sharp rise in the latter probability is possible if attackers manage to originate no more than 40% of the VMs requests in a given IaaS cloud.

Chapter 7
Summary and Conclusions

Because co-residency is a necessary first step to launching side channel attacks, this motivated this thesis to look into understanding the co-residency probability. As set forth in Section 1.2, this thesis aims to analyse and quantify the influence of cloud parameters (such as the number of host and users) on the co-residency probability under four commonly used *PA*s. These *PA*s are First Fit, Next Fit, Power Save and Random. By doing so, this thesis was able to identify the influential parameters' settings that reduce the co-residency probability in each *PA*. Reducing the attack surface for side channel attacks is, therefore, one outcome of reducing the co-residency probability.

This thesis achieved its aim through quantitative experimental simulation and analytical prediction. This approach consisted of four main steps:

(1) **Characterizing** the co-residency occurrence behavior in IaaS clouds using co-residency metrics (Chapter 3), followed by
(2) **Identifying** the four most influential cloud parameters (such as the number of hosts, clusters and users) affecting co-residency probability in four *PA*s. In order to do so, Chapter 4 quantified the influence of multiple cloud parameters on the co-residency probability. Then,
(3) **Simulation** experimentation to find the best settings of the most influential parameters that reduce the co-residency probability under four *PA*s (Chapter 5), finishing with
(4) **Analytical** estimation with the coexistence of different populations of attackers, to find the probability that a new co-residing VM belongs to an attacker (Chapter 6). These estimates helped to identify the best *PA*s that reduce the aforementioned probability.

The above steps were posed as research questions in Section 1.3. This chapter will revisit the research questions, summarizing the key findings and their implications in Section 7.1.

Section 7.2 draws some conclusions, followed by highlighting the limitations of this thesis in Section 7.3. Finally, Section 7.4 proposes potentially fruitful avenues for future research.

7.1 Summary

There are two hypotheses set forward in Section 1.3:

1. For a given *PA*, cloud parameters such as the number of hosts and users do not have the same influence on the co-residency probability in IaaS clouds.
2. For a given VM, there is a non-zero probability that a new co-residing VM belongs to an attacker for any of the four *PA*s considered.

An analysis of variance (ANOVA) test has been applied to the simulation estimates in Section 4.3.4. This allows the quantifying of the influence of eight cloud parameters and parameters interactions on the co-residency metrics under each *PA* (Section 4.5). This quantification showed that this influence varies with parameters, and, therefore, provided evidence to support the first hypothesis.

Further, the findings in Chapter 6 are based on an analytical estimation that seems to be in favour of the second hypothesis. The analytical estimation in Section 6.4.2 compared the probability that a new co-residing VM belongs to an attacker in four *PA*s over a wide range of α (i.e. attackers' VMs requests ratio). Given that attackers exist in the IaaS cloud, the analytical estimation results show that there is a non-zero probability that a new co-residing VM belongs to an attacker in all *PA*s.

The following is a brief summary of the key findings under each of the research questions.

1. How to characterise the co-residency occurrence behavior in IaaS clouds?

Following the description of the system and attack models in Section 3.2, four co-residency metrics characterizing the co-residency occurrence behaviour in IaaS clouds have been successfully defined in Chapter 3. Some of these characteristics include how likely a given VM *u* will be co-resided by another VM *v* (i.e. the co-residency probability), as well as how long does this co-residency take to occur.

These co-residency metrics were estimated using simulation to quantify the parameters' influence on the co-residency metrics. This quantification led to identifying the four most influential parameters and interactions on the co-residency metrics (Chapter 4).

These estimates also helped to find the best parameter settings that reduce the co-residency probability in four *PA*s (Chapter 5).

In addition, the co-residency metrics were used to derive analytical estimates of probabilities related to malicious co-residency (Chapter 6).

These metrics proved to be very useful in answering the research questions, and should also be useful to further research on co-residency in IaaS clouds.

2. For a given *PA*, what are the four most influential cloud parameters (such as the number of hosts, clusters and users) affecting the co-residency probability?

Due to the limited resources and time, this thesis focuses on the cloud parameters that have the most influence on the co-residency metrics. Therefore, an Influence Evaluation Strategy has been introduced (Section 4.3). This strategy statistically quantifies the influence on the co-residency metrics across a variety of likely cloud parameters' settings in four commonly used *PA*s. These *PA*s are First Fit, Next Fit, Power Save and Random (Appendix A provides a detailed description of these *PA*s). The strategy has applied Fractional Factorial design (Appendix B) to obtain reduced-size experiments (Section 4.3.3). Then, the strategy has used the *VMC* simulator to run these experiments to estimate the co-residency metrics across a variety of likely cloud parameters' settings in four *PA*s. Next, an Analysis of variance (ANOVA) test has been applied to the simulation estimates in Section 4.3.4. As a result, the strategy successfully identified the four most influential parameters and parameters interactions on the co-residency metrics under each *PA* (Section 4.5).

One of the most important findings in (Section 4.6) is that, out of many parameters that define the IaaS cloud environment, the number of hosts is the most influential parameter across the four *PA*s. The following are the four most influential parameters and two-parameter interactions on the co-residency probability. Number of hosts, user arrival rate, VM average lifetime and maximum host utilization were the four most influential parameters in First Fit and Power Save. The four most influential parameters in Next Fit and Random were the number of hosts, the interaction of the number of clusters and VMs per request parameters, user arrival rate and the interaction of the number of clusters and users' arrival rate parameters.

In addition, this thesis is the first to compare four *PA*s in terms of their impact on the co-residency probability and to identify that similarity exists between First Fit and Power

Save, as well as between Next Fit and Random.

The proposed Influence Evaluation Strategy is hoped to help researchers to identify the most influential parameters on the co-residency probability under different *PA*s.

3. For a given *PA*, which parameter settings reduce the co-residency probability?

Using the *VMC* simulator as a testbed, the four most influential parameters identified in Chapter 4 were used in controlled experiments in Chapter 5. These simulation experiments are aimed to explore how the most influential parameters' settings in four *PA*s could positively and negatively affect the co-residency metrics. In order to achieve this aim, these experiments estimated the co-residency metrics in four *PA*s under a wide range of likely settings for publicly accessible IaaS clouds (Section 5.2).

Next, Pearson's correlation analysis has been applied to study the correlation between these parameters and the co-residency metrics. This analysis helped in identifying the parameters' settings that were able to reduce the co-residency probability in each *PA* (see Table 5.17). Based on this finding, Section 5.4 presents evidence that VMs hosted in IaaS clouds with a higher number of hosts are less likely to exhibit co-residency.

Further, using Next Fit in larger IaaS clouds has been shown to reduce effectively, and even eliminate, the co-residency probability. In addition, the four *PA*s have been compared in their ability to reduce the co-residency probability. For instance, VMs in IaaS clouds that use Next Fit or Random are more resilient to the reception of co-resident VMs compared to when First Fit or Power Save are used.

4. For a given VM, what is the probability that a new co-residing VM belongs to an attacker?

The risk of side channel attacks is magnified enormously if an honest VM is co-resided by an attacker. Therefore, this research question investigated reducing the probability that the next co-residing VM belongs to an attacker (i.e. the malicious co-residency probability). Chapter 6 defined two metrics (i.e. the *MCP* and *AFL*) that describe probabilities related to malicious co-residency and also take into account the attackers' VMs requests ratio α. This thesis is the first to derive two approximate analytical estimates of probabilities related to malicious co-residency in Section 6.2.

Then, analytical estimates of the *MCP* and *AFL* have been compared with experimental estimates (i.e. using the *VMC* simulator) in four *PA*s under an α value of 0.10. The results in (Table 6.4, Table 6.5, Table 6.6, and Table 6.7) show the analytical estimates, the simulation estimates and the corresponding percentage differences. About 75% and 100% of the obtained analytical estimates of the *MCP* and *AFL,* respectively*,* had percentage differences less than 15% in the four *PA*s. Moreover, the mean percentage differences are 10.31% and 2.31% for the *MCP* and *AFL*, respectively. On the other hand, the *MCP* was overestimated in First Fit and Power Save as shown in the percentage difference that increased to levels that were pre-defined as not being adequate (Section 6.3). Therefore, the derived analytical estimates were shown to agree with the experimental estimates in Section 6.4.1.

Further, Section 6.4.2 used the derived analytical estimates to compare First Fit, Next Fit, Power Save and Random over a wide range of α values. By comparing the *PA*s as α varies between 0 to 0.99 (Table 6.8 and Table 6.9), the probability (with 95% confidence intervals) that an honest VM *u* encounters a malicious co-residency hit at least once during its lifetime (i.e. the *MCP*) is between 0.197 to 0.376 in Next Fit, compared to 0.270 to 0.514 in Random, 0.490 to 0.862 in First Fit and 0.487 to 0.860 in Power Save. Thus, the aforementioned findings demonstrated that VMs hosted in IaaS clouds that use Next Fit or Random are less likely to receive co-resident attacker VMs compared to when First Fit or Power Save are used. The findings also suggest that the right choice of *PA*s can reduce the probability of being co-resided by attackers' VMs, which can reduce the attack surface for side channel attacks. However, an interesting finding in Section 6.4.2 shows that a sharp rise in the latter probability is possible if attackers manage to originate no more than 40% of the VMs requests in a given IaaS cloud.

7.2 Conclusion

With co-residency being inevitable in public IaaS clouds, adverse consequences of side channels, brought by co-residency, are shown to affect the VMs security in multi-tenant public IaaS clouds. Because co-residency is a necessary first step to launching side channel attacks, this motivated this thesis to look into understanding the co-residency probability. Based on the summary in the previous section, this thesis successfully accomplished its aim by analysing and quantifying the influence of cloud parameters on the co-residency probability under four commonly used *PA*s. These *PA*s are First Fit, Next Fit, Power Save

and Random. Out of many parameters that define the IaaS cloud environment, the number of hosts was the most influential parameter across the four *PA*s. In addition, the findings of this thesis shed new light on the conditions under which the co-residency probability varies. For instance, the co-residency probability has been shown to decrease as the number of hosts increases in IaaS clouds.

After identifying the most influential parameters, this thesis has demonstrated that determining and employing the appropriate parameters' settings in a given *PA* can effectively reduce the co-residency probability in public IaaS clouds. Table 5.17 lists the best parameters' settings in four *PA*s that reduced the co-residency probability.

The work presented in this thesis is a plausible blueprint for IaaS cloud providers to consider co-residency reduction as an important selection factor for *PA*s and cloud settings (such as the number of hosts). Reducing the residency probability should complement the available countermeasures to side channel attacks (Section 2.3.3) by reducing the attack surface for side channel attacks.

The derived analytical estimates may also be useful for IaaS cloud providers and users for estimating the co-residency probability in various IaaS cloud settings and *PA*s.

7.3 Limitations

Since this work has been an exploratory venture into a little-chartered territory, a number of assumptions had to be made to answer the research questions. Therefore, this thesis inevitably has some limitations, the most significant of which are discussed in this section.

Analyzing and quantifying the influence of cloud parameters on the co-residency probability has been based on an attack scenario in Section 3.2. The attack scenario makes assumptions about how an attacker places malicious VMs, and, therefore, the analysis may be invalidated if these assumptions fail to hold. Given a victim VM u and an attacker VM v, one of the most basic assumptions is that v co-resides with u during the latter's lifetime. This assumption is supported by a demonstrated co-residing technique (see Section 2.3.1.2) that target specific and existing VMs in public IaaS clouds, but may not hold for every type of technique. One particular type of co-residing technique for which it may not hold is when the attacker places many replicas of VM v in random hosts hoping that VM u becomes co-resident in a later stage. Achieving co-residency using this technique might be possible for organized attackers

with plentiful resources (e.g., organization-sponsored attackers). However, this co-residing technique falls outside the scope of this thesis. Preventing this technique may require the IaaS cloud provider to monitor and limit attackers' ability to request a vast number of VMs.
Another fundamental assumption made is that the used co-residing technique follows an attack model where an attacker relies ultimately on the *PA*'s decision when attempting to co-reside with victim VMs. Hence, this thesis does not consider a situation where an attacker is an insider (e.g. a system administrator) who is capable of enforcing VM placement to co-reside with victim VMs. This type of attack, which involves cloud insiders, is shown to be feasible in the real-world [81]. Therefore, the aforementioned limitations suggest that the outcome of this thesis is not applicable to all kinds of attacks.
It is important to re-emphasise the fact that co-residency reduction does not prevent side channels; it instead aims to make co-residing with VMs in public IaaS clouds more difficult.

7.4 Future Work

The possibility of employing the right cloud parameters' settings in four commonly used *PA*s to reduce the co-residency probability has been demonstrated in this thesis. Therefore, several future directions for research emerge. For instance, analysing and quantifying the influence of various cloud parameters on the co-residency probability in more *PA*s.
In addition, the co-residency probability can be used as a useful benchmark for comparing public IaaS clouds based on how their cloud settings and *PA* reduce the co-residency probability.
Another interesting line of research would be to design *PA*s that reduce the co-residency probability and also take into account other important aspects, such as performance and energy consumptions. This kind of *PA*s might prove to be very useful in practice. A very recent and promising attempt was made in this context by [10], formalizing a secure *PA* that prevents a specific type of co-residency (see Section 2.3.3.4).
Moreover, an increasing number of publications have compared *PA*s in several aspects. Such aspects include cost reduction [37], [48], [49] and performance and energy consumptions [40], [55], [58], [99]. This thesis is the first to compare four *PA*s in terms of their impact on the co-residency probability and to identify that a similarity exists between First Fit and Power Save, as well as between Next Fit and Random. These findings open an interesting area for future research that involves comparing more *PA*s in terms of how much they are

likely to reduce the co-residency probability.

Another interesting line of results from the previous *PA*s comparison would be to establish lower and upper bounds on the trade-off between performance and resilience to co-residency for each *PA*. More precisely, finding a lower bound on each *PA*'s performance and an upper bound on the expected probability of co-residency. This would be very helpful, within the context of this thesis, to identify the cost of using each *PA* to secure against side channel attacks.

Bibliography

[1] A. Alabdulhafez and P. Ezhilchelvan, "Experimenting on Virtual Machines Co-residency in the Cloud," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14*, 2014, pp. 363–366.

[2] A. Alabdulhafez and P. Ezhilchelvan, "Analyzing the Success Rate of Virtual Machines Co-residency in the Cloud," in *Proceedings of the 6th Saudi Scientific International Conference (SIC)*, 2012, pp. 164–168.

[3] H. Aljahdali, P. Townend, and J. Xu, "Enhancing Multi-tenancy Security in the Cloud IaaS Model Over Public Deployment," in *Proceedings of the IEEE Seventh International Symposium on Service-Oriented System Engineering*, 2013, pp. 385–390.

[4] Amazon EC2, "AWS | Amazon Elastic Compute Cloud (EC2)," 2014. [Online]. Available: http://aws.amazon.com/ec2/. [Accessed: 23-Oct-2014].

[5] Amazon EC2, "FAQs: How Many Instances Can I Run in Amazon EC2?," 2014. [Online]. Available: http://aws.amazon.com/ec2/faqs/#How_many_instances_can_I_run_in_Amazon_EC2. [Accessed: 29-Oct-2014].

[6] J. Araujo, R. Matos, V. Alves, P. Maciel, V. Souza, R. Matias, and K. Trivedi, "Software Aging in the Eucalyptus Cloud Computing Infrastructure," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 10, no. 1, pp. 1–22, Jan. 2014.

[7] S. F. Arnold, *Design of Experiments with MINITAB*. The American Statistician, 2006.

[8] A. Aviram, S. Hu, B. Ford, and R. Gummadi, "Determinating Timing Channels in Compute Clouds," in *Proceedings of the ACM Workshop on Cloud Computing Security - CCSW '10*, 2010, pp. 103–108.

[9] P. Ayers, "Securing and Controlling Data in the Cloud," *Computer Fraud and Security*, vol. 2012, no. 11, pp. 16–20, 2012.

[10] Y. Azar, S. Kamara, I. Menache, M. Raykova, and B. Shepard, "Co-Location-Resistant Clouds," in *Proceedings of the 6th ACM Workshop on Cloud Computing Security - CCSW '14*, 2014, pp. 9–20.

[11] M. B. Baig, C. Fitzsimons, S. Balasubramanian, R. Sion, and D. E. Porter, "CloudFlow: Cloud-wide Policy Enforcement Using Fast VM Introspection," in *Proceedings of the IEEE International Conference on Cloud Engineering*, 2014, pp. 159–164.

[12] J. Banks, C. John S., B. L. Nelson, and D. M. Nicol, *Discrete-event System Simulation*. Prentice Hall, 2010.

[13] A. Bates, B. Mood, J. Pletcher, H. Pruse, M. Valafar, and K. Butler, "Detecting Co-residency with Active Traffic Analysis Techniques," in *Proceedings of the ACM Workshop on Cloud Computing Security Workshop - CCSW '12*, 2012, pp. 1–12.

[14] G. van Belle, *Statistical Rules of Thumb*. John Wiley & Sons, 2011.

[15] G. E. P. Box, J. S. Hunter, and W. G. Hunter, *Statistics for Experimenters: Design, Innovation, and Discovery*. Wiley-Interscience, 2005.

[16] J. Brodkin, "VMware Confirms Source Code Leak, LulzSec-affiliated Hacker Claims Credit | Ars Technica," 2012. [Online]. Available: http://arstechnica.com/business/2012/04/vmware-confirms-source-code-leak-lulzsec-affiliated-hacker-claims-credit/. [Accessed: 19-Oct-2014].

[17] S. Bugiel, S. Nürnberger, T. Pöppelmann, A.-R. Sadeghi, and T. Schneider, "AmazonIA: When Elasticity Snaps Back," in *Proceedings of the 18th ACM Conference on Computer and Communications Security - CCS '11*, 2011, pp. 389–400.

[18] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[19] H. Casanova, "Simgrid: A Toolkit for the Simulation of Application Scheduling," in *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2001, pp. 430–437.

[20] R. Chiang, S. Rajasekaran, N. Zhang, and H. H. Huang, "Swiper: Exploiting Virtual Machine Vulnerability in Third-Party Clouds with Competition for I/O Resources," *IEEE Transactions on Parallel and Distributed Systems*, no. 1, pp. 1–10, 2014.

[21] D. Chisnall, *The Definitive Guide to the Xen Hypervisor*. Prentice Hall, 2008.

[22] H. Coolican and P. L. in P. H. Coolican, *Research Methods and Statistics in Psychology, Sixth Edition*. Psychology Press, 2014.

[23] CVE, "CVE-2007-4993 : pygrub (tools/pygrub/src/GrubConf.py) in Xen 3.0.3, When Booting a Guest Domain, Allows Local Users with Elevated Privileges to Execute Arbitrary Commands," 2007. [Online]. Available: http://www.cvedetails.com/cve/CVE-2007-4993/. [Accessed: 19-Oct-2014].

[24] CVE, "CVE-2007-5497 : Multiple Integer Overflows in libext2fs in e2fsprogs Before 1.40.3 Allow User-assisted Remote Attackers to Execute Code," 2007. [Online]. Available: http://www.cvedetails.com/cve/CVE-2007-5497/. [Accessed: 19-Oct-2014].

[25] CVE, “CVE-2010-2240 - The do_anonymous_page function in mm/memory.c in the Linux Kernel Before 2.6.27.52, 2.6.32.x,” 2010. [Online]. Available: http://cve.circl.lu/cve/CVE-2010-2240. [Accessed: 19-Oct-2014].

[26] C. P. Dancey and J. Reidy, *Statistics Without Maths for Psychology*. Pearson/Prentice Hall, 2007.

[27] C. L. Dumitrescu and I. Foster, “GangSim: A Simulator for Grid Scheduling Studies,” in *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid*, 2005, vol. 2, pp. 1151–1158.

[28] A. EC2, “AWS | Amazon EC2 Dedicated Instances,” 2014. [Online]. Available: http://aws.amazon.com/ec2/purchasing-options/dedicated-instances/. [Accessed: 09-Nov-2014].

[29] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.

[30] S. K. Garg and R. Buyya, “NetworkCloudSim: Modelling Parallel Applications in Cloud Simulations,” in *Proceedings of the Fourth IEEE International Conference on Utility and Cloud Computing*, 2011, pp. 105–113.

[31] M. Godfrey and M. Zulkernine, “A Server-Side Solution to Cache-Based Side-Channel Attacks in the Cloud,” in *Proceedings of the IEEE Sixth International Conference on Cloud Computing*, 2013, pp. 163–170.

[32] I. Gorka, S. I. Mehmet, E. Thomas, and B. Sunar, “Wait a Minute! A Fast, Cross-VM Attack on AES,” in *Proceedings of the 17th International Symposium on Research in Attacks, Intrusions and Defenses*, 2014, pp. 299–319.

[33] R. F. Gunst and R. L. Mason, “Fractional Factorial Design,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 1, no. 2, pp. 234–244, 2009.

[34] J. Gustedt, E. Jeannot, and M. Quinson, “Experimental Methodologies For Large-Scale Systems: A Survey,” *Parallel Processing Letters*, vol. 19, no. 03, pp. 399–418, Sep. 2009.

[35] J. O. Henriksen, “An Introduction to SLX,” in *Proceedings of the 29th Conference on Winter Simulation - WSC ’97*, 1997, pp. 559–566.

[36] W.-M. Hu, “Reducing Timing Channels with Fuzzy Time,” in *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, 1991, pp. 8–20.

[37] C. Hyser, B. Mckee, R. Gardner, and B. Watson, *Autonomic Virtual Machine Placement in the Data Center*. HP Laboratories, 2007.

[38] T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*. Boston, MA: Springer US, 2009.

[39] Y. Jararweh, Z. Alshara, M. Jarrah, M. Kharbutli, and M. N. Alsaleh, “TeachCloud: A Cloud Computing Educational Toolkit,” *International Journal of Cloud Computing*, vol. Volume 2, no. 2, pp. 237–257, 2013.

[40] Jenn-Wei Lin and Chien-Hung Chen, “Interference-aware Virtual Machine Placement in Cloud Computing Systems,” in *Proceedings of the International Conference on Computer & Information Science (ICCIS)*, 2012, vol. 2, pp. 598–603.

[41] E. Keller, J. Szefer, J. Rexford, and R. B. Lee, “NoHype: Virtualized Cloud Infrastructure Without the Virtualization,” *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 350–361, Jun. 2010.

[42] G. Keramidas, A. Antonopoulos, D. N. Serpanos, and S. Kaxiras, “Non Deterministic Caches: A Simple and Effective Defense Against Side Channel Attacks,” *Design Automation for Embedded Systems*, vol. 12, no. 3, pp. 221–230, Apr. 2008.

[43] E. Kijsipongse and S. Vannarat, “Autonomic Resource Provisioning in Rocks Clusters Using Eucalyptus Cloud Computing,” in *Proceedings of the International Conference on Management of Emergent Digital EcoSystems - MEDES ’10*, 2010, pp. 61–66.

[44] D. Kliazovich, P. Bouvry, Y. Audzevich, and S. U. Khan, “GreenCloud: A Packet-Level Simulator of Energy-Aware Cloud Computing Data Centers,” in *Proceedings of the IEEE Global Telecommunications Conference GLOBECOM 2010*, 2010, pp. 1–5.

[45] W. Kreutzer, J. Hopkins, and M. van Mierlo, “SimJAVA: Framework for Modeling Queueing Networks in Java,” in *Proceedings of the 29th Conference on Winter Simulation - WSC ’97*, 1997, pp. 483–488.

[46] G. Kurian, O. Khan, and S. Devadas, “The Locality-aware Adaptive Cache Coherence Protocol,” in *Proceedings of the 40th Annual International Symposium on Computer Architecture - ISCA ’13*, 2013, vol. 41, no. 3, pp. 523–534.

[47] B. W. Lampson, “A Note on the Confinement Problem,” *Communications of the ACM*, vol. 16, no. 10, pp. 613–615, Oct. 1973.

[48] K. Le, R. Bianchini, J. Zhang, Y. Jaluria, J. Meng, and T. D. Nguyen, “Reducing Electricity Cost Through Virtual Machine Placement in High Performance Computing Clouds,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC ’11*, 2011, pp. 66–72.

[49] B. Li, J. Li, J. Huai, T. Wo, Q. Li, and L. Zhong, “EnaCloud: An Energy-Saving Application Live Placement Approach for Cloud Computing Environments,” in *Proceedings of the IEEE International Conference on Cloud Computing*, 2009, pp. 17–24.

[50] P. Li, D. Gao, and M. Reiter, “Mitigating Access-Driven Timing Channels in Clouds Using StopWatch,” in *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2013, pp. 1–12.

[51] S.-H. Lim, B. Sharma, G. Nam, E. K. Kim, and C. R. Das, “MDCSim: A Multi-tier Data Center Simulation Platform,” in *Proceedings of the IEEE International Conference on Cluster Computing and Workshops*, 2009, pp. 1–9.

[52] F. Liu, L. Ren, and H. Bai, “Mitigating Cross-VM Side Channel Attack on Multiple Tenants Cloud Platform,” *Journal of Computers*, vol. 9, no. 4, pp. 1005–1013, Apr. 2014.

[53] H. Liu, “A New Form of DOS Attack in a Cloud and its Avoidance Mechanism,” in *Proceedings of the ACM on Cloud Computing Security Workshop - CCSW ’10*, 2010, pp. 65–76.

[54] J. Liu, F. Zhao, X. Liu, and W. He, “Challenges Towards Elastic Power Management in Internet Data Centers,” in *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems Workshops*, 2009, pp. 65–72.

[55] X.-F. Liu, Z.-H. Zhan, K.-J. Du, and W.-N. Chen, “Energy Aware Virtual Machine Placement Scheduling in Cloud Computing Based on Ant Colony Optimization Approach,” in *Proceedings of the Conference on Genetic and Evolutionary Computation - GECCO ’14*, 2014, pp. 41–48.

[56] P. M. Mell and T. Grance, “The NIST Definition of Cloud Computing,” National Institute of Standards & Technology, Sep. 2011.

[57] Microsoft, “Azure: Microsoft’s Cloud Platform,” 2014. [Online]. Available: http://azure.microsoft.com/en-us/. [Accessed: 23-Oct-2014].

[58] K. Mills, J. Filliben, and C. Dabrowski, “Comparing VM-Placement Algorithms for On-Demand Clouds,” in *Proceedings of the IEEE Third International Conference on Cloud Computing Technology and Science*, 2011, pp. 91–98.

[59] K. Mills, J. Filliben, and C. Dabrowski, “An Efficient Sensitivity Analysis Method for Large Cloud Simulations,” in *Proceedings of the IEEE 4th International Conference on Cloud Computing*, 2011, pp. 724–731.

[60] D. Milojičić, I. M. Llorente, and R. S. Montero, “OpenNebula: A Cloud Management Tool,” *IEEE Internet Computing*, vol. 15, no. 2, pp. 11–14, Mar. 2011.

[61] S. Naicken, A. Basu, B. Livingston, and S. Rodhetbhai, “Towards Yet Another Peer-to-Peer Simulator,” in *Proceedings of The Fourth International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks*, 2006, pp. 37–47.

[62] A. Nathani, S. Chaudhary, and G. Somani, “Policy Based Resource Allocation in IaaS Cloud,” *Future Generation Computer Systems*, vol. 28, no. 1, pp. 94–103, Jan. 2012.

[63] Neovise, “Public, Private and Hybrid Clouds - When, Why and How They are Really Used,” 2013. [Online]. Available: http://www.virtustream.com/company/buzz/press-releases/neovise-research-report. [Accessed: 13-Nov-2014].

[64] A. Nuñez, J. L. Vázquez-Poletti, A. C. Caminero, J. Carretero, and I. M. Llorente, "Design of a New Cloud Computing Simulation Platform," in *Proceedings of the International Conference on Computational Science and its Applications*, 2011, pp. 582–593.

[65] A. Núñez, J. Fernández, R. Filgueira, F. García, and J. Carretero, "SIMCAN: A Flexible, Scalable and Expandable Simulation Platform for Modelling and Simulating Distributed Architectures and Applications," *Simulation Modelling Practice and Theory*, vol. 20, no. 1, pp. 12–32, Jan. 2012.

[66] A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero, and I. M. Llorente, "iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator," *Journal of Grid Computing*, vol. 10, no. 1, pp. 185–209, Apr. 2012.

[67] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-Source Cloud-Computing System," in *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009, pp. 124–131.

[68] S. Ostermann, K. Plankensteiner, D. Bodner, G. Kraler, and R. Prodan, "Integration of an Event-Based Simulation Framework into a Scientific Workflow Execution Environment for Grids and Clouds.," in *Proceedings of the Towards a Service-Based Internet - 4th European Conference*, 2011, pp. 1–13.

[69] S. Ostermann, K. Plankensteiner, R. Prodan, and T. Fahringer, "GroudSim: An Event-based Simulation Framework for Computational Grids and Clouds," in *Proceedings of the Conference on Parallel Processing*, 2010, pp. 305–313.

[70] D. A. Osvik, A. Shamir, and E. Tromer, "Cache Attacks and Countermeasures: The Case of AES," in *Proceedings of the Cryptographers' Track at the RSA Conference on Topics in Cryptology*, 2006, pp. 1–20.

[71] D. Page, "Partitioned Cache Architecture as a Side-Channel Defence Mechanism," *IACR Cryptology ePrint Archive*, vol. 280, no. 8, pp. 10–16, 2005.

[72] D. Page, "Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel," *IACR Cryptology ePrint Archive*, vol. 190, no. 2, pp. 14–22, 2002.

[73] D. Page, "Defending Against Cache-based Side-channel Attacks," *Information Security Technical Report*, vol. 8, no. 1, pp. 30–44, Mar. 2003.

[74] V. Paxson and S. Floyd, "Why We Don't Know How to Simulate the Internet," in *Proceedings of the 29th Conference on Winter Simulation - WSC '97*, 1997, pp. 1037–1044.

[75] C. Percival, "Cache Missing for Fun and Profit," in *Proceedings of the BSDCan*, 2005, pp. 1–12.

[76] N. Pitropakis, A. Pikrakis, and C. Lambrinoudakis, “Behaviour Reflects Personality: Detecting Co-residence Attacks on Xen-based Cloud Environments,” *International Journal of Information Security*, vol. 1, no. 1, pp. 1–7, Aug. 2014.

[77] Rackspace, “Rackspace Cloud Company,” 2014. [Online]. Available: http://www.rackspace.com/. [Accessed: 30-Oct-2014].

[78] A. Reed, C. Rezek, and P. Simmonds, *Security Guidance for Critical Areas of Focus in Cloud Computing V3.0*. Cloud Security Alliance, 2011.

[79] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, “Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds,” in *Proceedings of the 16th ACM Conference on Computer and Communications Security - CCS ’09*, 2009, pp. 199–212.

[80] T. J. Rivlin, *An Introduction to the Approximation of Functions*. Courier Dover Publications, 1981.

[81] F. Rocha and M. Correia, “Lucy in the Sky Without Diamonds: Stealing Confidential Data in the Cloud,” in *Proceedings of the IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2011, pp. 129–134.

[82] C. L. Sabharwal, “Thinking in Java,” *IEEE Potentials*, vol. 17, no. 3, pp. 33–37, 1998.

[83] J. Sauro and J. R. Lewis, *Quantifying the User Experience: Practical Statistics for User Research*. Elsevier, 2012.

[84] W. Sellami, H. H. Kacem, and A. H. Kacem, “Towards a Multi-tenancy Aware Cloud Service Composition,” in *Proceedings of the 28th International Conference on Advanced Information Networking and Applications Workshops*, 2014, pp. 404–409.

[85] P. Sempolinski and D. Thain, “A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus,” in *Proceedings of the IEEE Second International Conference on Cloud Computing Technology and Science*, 2010, pp. 417–426.

[86] SimGrid Documentation, “SimGrid Simulator: VMs,” 2014. [Online]. Available: http://simgrid.gforge.inria.fr/simgrid/latest/doc/group__msg__VMs.html. [Accessed: 31-Oct-2014].

[87] D. X. Song, D. Wagner, and X. Tian, “Timing Analysis of Keystrokes and Timing Attacks on SSH,” in *Proceedings of the 10th Conference on USENIX Security Symposium*, 2001, pp. 250–258.

[88] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien, “The MicroGrid: a Scientific Tool for Modeling Computational Grids,” in *Proceedings of the IEEE Supercomputing Conference (SC’2000)*, 2000, pp. 127–141.

[89] B. B. Stone and A. Vance, “Companies Slowly Join Cloud-Computing,” *The New York Times*, 18-Apr-2010.

[90] N. Sudarsanam and D. D. Frey, “Using Ensemble Techniques to Advance Adaptive One-factor-at-a-time Experimentation,” *Quality and Reliability Engineering International*, vol. 27, no. 7, pp. 947–957, Nov. 2011.

[91] J. Szefer, E. Keller, R. B. Lee, and J. Rexford, “Eliminating the Hypervisor Attack Surface for a More Secure Cloud,” in *Proceedings of the 18th ACM Conference on Computer and Communications Security - CCS ’11*, 2011, pp. 401–412.

[92] V. Varadarajan, T. Kooburat, B. Farley, T. Ristenpart, and M. M. Swift, “Resource-freeing Attacks: Improve Your Cloud Performance (at Your Neighbor’s Expense),” in *Proceedings of the ACM Conference on Computer and Communications Security - CCS ’12*, 2012, pp. 281–292.

[93] B. C. Vattikonda, S. Das, and H. Shacham, “Eliminating Fine Grained Timers in Xen,” in *Proceedings of the 3rd ACM Cloud Computing Security workshop - CCSW ’11*, 2011, pp. 41–46.

[94] VMware, “VMSA-2008-0008,” 2008. [Online]. Available: http://www.vmware.com/security/advisories/VMSA-2008-0008. [Accessed: 19-Oct-2014].

[95] J. Wang and M. N. Huhns, “Using Simulations to Assess the Stability and Capacity of Cloud Computing Systems,” in *Proceedings of the 48th ACM Annual Southeast Regional Conference - ACM SE ’10*, 2010, pp. 9–19.

[96] B. Wickremasinghe, R. N. Calheiros, and R. Buyya, “CloudAnalyst: A CloudSim-Based Visual Modeller for Analysing Cloud Computing Environments and Applications,” in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications*, 2010, pp. 446–452.

[97] Z. Wu, Z. Xu, and H. Wang, “Whispers in the Hyper-Space: High-Bandwidth and Reliable Covert Channel Attacks Inside the Cloud,” *IEEE/ACM Transactions on Networking*, vol. PP, no. 99, pp. 1–10, 2014.

[98] F. Wuhib, R. Stadler, and H. Lindgren, “Dynamic Resource Allocation with Management Objectives: Implementation for an OpenStack Cloud,” in *Proceedings of the 8th International Conference on Network and Service Management*, 2012, pp. 309–315.

[99] J. Xu and J. A. B. Fortes, “Multi-Objective Virtual Machine Placement in Virtualized Data Center Environments,” in *Proceedings of the IEEE/ACM International Conference on Green Computing and Communications and International Conference on Cyber, Physical and Social Computing*, 2010, pp. 179–188.

[100] S. Yu, X. Gui, and J. Lin, “An Approach with Two-stage Mode to Detect Cache-based Side Channel Attacks,” in *Proceedings of the International Conference on Information Networking 2013 (ICOIN)*, 2013, pp. 186–191.

[101] M. Zaigham, *Cloud Computing: Challenges, Limitations and R&D Solutions*. Cham: Springer International Publishing, 2014.

[102] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter, "HomeAlone: Co-residency Detection in the Cloud via Side-Channel Analysis," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2011, pp. 313–328.

[103] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-VM Side Channels and Their Use to Extract Private Keys," in *Proceedings of the ACM Conference on Computer and Communications Security - CCS '12*, 2012, pp. 305–316.

[104] Y. Zhang and M. K. Reiter, "Düppel: Retrofitting Commodity Operating Systems to Mitigate Cache Side Channels in the Cloud," in *Proceedings of the ACM Conference on Computer & Communications Security - CCS '13*, 2013, pp. 827–838.

Appendix A

VMC Simulator Implementation

A.1 Definition

This appendix describes how the *VMC* simulator implements the system model and the four *PA*s used in this thesis (Section 3.2). The following are important preliminary definitions and assumptions:

- The number of clusters in the system model is denoted as C
- Each cluster is assigned a unique identifier i, such that $0 < i \leq C$
- Each cluster has a number of hosts ω
- Each host is assigned a unique identifier j, such that $0 < j \leq \omega$
- A host is *available* if it has an available space to accommodate a new VM
- A host is *full* if it cannot accommodate a new VM
- A cluster is *available* if it contains at least one available host
- A cluster is *full* if it does not contain any available host
- The *VMC* simulator implements the system model as follows:
    - Clusters are ordered by their identifiers from lowest to highest
    - Hosts in cluster i are ordered by their identifiers from lowest to highest
    - The lowest cluster/host refers to the cluster/host with the lowest identifier number

In addition, the attackers VMs requests rate α can be specified to the *VMC* simulator in order to estimate probabilities related to malicious co-residency (see Chapter 6).

Further, the *VMC* defines the following 36 input parameters to describe the system model in this thesis (Table A.1):

ID	Parameters	Description
1.	Number of Clusters	How many Clusters in the simulation
2.	Number of Hosts of Type ***H1***	How many Hosts of each type to be created in the simulation. Hosts will be distributed randomly into clusters.
3.	Number of Hosts of Type ***H2***	
4.	Number of Hosts of Type ***H3***	
5.	Number of Hosts of Type ***H4***	
6.	Number of Hosts of Type ***H5***	
7.	Max Utilization for Host Type ***H1***	A Host is Full when the hosted VMs usage of the host's resources (CPU, memory and storage) reaches the Max Host Utilization percentage.
8.	Max Utilization for Host Type ***H2***	
9.	Max Utilization for Host Type ***H3***	
10.	Max Utilization for Host Type ***H4***	
11.	Max Utilization for Host Type ***H5***	
12.	Users' Arrival Rate Of Type ***U1***	Average number of new users of each type to be created every time unit
13.	Users' Arrival Rate Of Type ***U2***	
14.	Users' Arrival Rate Of Type ***U3***	
15.	Users' Arrival Rate Of Type ***U4***	
16.	Users' Arrival Rate Of Type ***U5***	
17.	Maximum Number of Users of Type ***U1***	The maximum number of users that can run in the simulated cloud simultaneously.
18.	Maximum Number of Users of Type ***U2***	
19.	Maximum Number of Users of Type ***U3***	
20.	Maximum Number of Users of Type ***U4***	

21.	Maximum Number of Users of Type ***U5***	
22.	Max Parallel VMs of User Type ***U1***	The maximum number of concurrent VMs a single user can run.
23.	Max Parallel VMs of User Type ***U2***	
24.	Max Parallel VMs of User Type ***U3***	
25.	Max Parallel VMs of User Type ***U4***	
26.	Max Parallel VMs of User Type ***U5***	
27.	***X_SMALL*** VM Average Lifetime	How long a user (on average) holds his running VM before terminating it (in time units)
28.	***SMALL*** VM Average Lifetime	
29.	***MEDIUM*** VM Average Lifetime	
30.	***LARGE*** VM Average Lifetime	
31.	***X_LARGE*** VM Average Lifetime	
32.	VMs per User Request Rate for User Type ***U1***	How many new VM(s) to be created in each new VMs request (on average). The number of VMs per request must be less than or equal to the Max Parallel VMs per User parameter.
33.	VMs per User Request Rate for User Type ***U2***	
34.	VMs per User Request Rate for User Type ***U3***	
35.	VMs per User Request Rate for User Type ***U4***	
36.	VMs per User Request Rate for User Type ***U5***	

Table A.1 The input parameters that define the *VMC* simulator.

In addition, the *VMC* produces the simulation outputs in an MS Excel file that contains five sheets in which each shows different statistical information. In addition, a log file is generated which displays the details of all the simulations actions and events in a text-based format (Figure A.1).

```
|(15) Clusters Created
 (5000) Hosts Created: Types /
=============>>> H1(265) H2(247) H3(504) H4(505) H5(3479)
 (3600) Minutes Created
Simulator Started
------------------------------------------------------------
In Minute (0):
Updating VMs Co-Location window time.............
Performing already scheduled actions on this minute
No scheduled actions found on this minute
+ + + +
Created Users  (3) - User Details:
User type: PU - User ID (0)
Action performed: ID (0) - User_ID (0) - Type (Create_VM) - VM Type: SMALL - VM_ID (0) Total of (2) VMs - Cluster ID (0) - Host ID (17)
Action performed: ID (0) - User_ID (0) - Type (Create_VM) - VM Type: SMALL - VM_ID (1) Total of (2) VMs - Cluster ID (1) - Host ID (27)
Action scheduled: ID (3) - User_ID (0) - Type (Create_VM) will be perfomed on Minute #(767)
User type: PU - User ID (1)
Action performed: ID (4) - User_ID (1) - Type (Create_VM) - VM Type: SMALL - VM_ID (2) Total of (1) VMs - Cluster ID (2) - Host ID (15)
Action scheduled: ID (6) - User_ID (1) - Type (Create_VM) will be perfomed on Minute #(767)
User type: PU - User ID (2)
Action performed: ID (7) - User_ID (2) - Type (Create_VM) - VM Type: SMALL - VM_ID (3) Total of (1) VMs - Cluster ID (3) - Host ID (12)
Action scheduled: ID (9) - User_ID (2) - Type (Create_VM) will be perfomed on Minute #(767)
Total of (3) new users  created
------------------------------------------------------------
In Minute (1):
Updating VMs Co-Location window time.............
Performing already scheduled actions on this minute
No scheduled actions found on this minute
+ + + +
Created Users  (3) - User Details:
User type: MS - User ID (3)
Action performed: ID (10) - User_ID (3) - Type (Create_VM) - VM Type: LARGE - VM_ID (4) Total of (2) VMs - Cluster ID (4) - Host ID (1)
Action performed: ID (10) - User_ID (3) - Type (Create_VM) - VM Type: LARGE - VM_ID (5) Total of (2) VMs - Cluster ID (5) - Host ID (4)
Action scheduled: ID (13) - User_ID (3) - Type (Create_VM) will be perfomed on Minute #(546)
User type: PS - User ID (4)
Action performed: ID (14) - User_ID (4) - Type (Create_VM) - VM Type: LARGE - VM_ID (6) Total of (2) VMs - Cluster ID (6) - Host ID (7)
Action performed: ID (14) - User_ID (4) - Type (Create_VM) - VM Type: LARGE - VM_ID (7) Total of (2) VMs - Cluster ID (7) - Host ID (20)
Action scheduled: ID (17) - User_ID (4) - Type (Create_VM) will be perfomed on Minute #(2572)
User type: PU - User ID (5)
Action performed: ID (18) - User_ID (5) - Type (Create_VM) - VM Type: SMALL - VM_ID (8) Total of (2) VMs - Cluster ID (8) - Host ID (5)
Action performed: ID (18) - User_ID (5) - Type (Create_VM) - VM Type: SMALL - VM_ID (9) Total of (2) VMs - Cluster ID (9) - Host ID (3)
Action scheduled: ID (21) - User_ID (5) - Type (Create_VM) will be perfomed on Minute #(2572)
Total of (3) new users  created
------------------------------------------------------------
In Minute (2):
Updating VMs Co-Location window time.............
Performing already scheduled actions on this minute
No scheduled actions found on this minute
+ + + +
Created Users  (3) - User Details:
User type: PU - User ID (6)
```

non-adv VMs ONLY

1.827143

	VMs Colocated Times	0	0	0.000736	0.278099	0.242987	0.091605	0.192428	0.250736	0.171538	0.180757	18098		0.958124		Durations (L) between Co-residency Hits (k)		Averages Average number of Co-residency Hits (k)	1.408886			

VM ID	Total Colocated	0-10 %	11-20 %	21-30 %	31-40 %	41-50 %	51-60 %	61-70 %	71-80 %	81-90 %	91-100 %	Colocated by adv	Order of 1st Adv Hit	% of Life Time Until 1st Adv Hit	L1	L2	L3	L4	L5	L6	L7
0	2	0	0	0	0	1	0	0	0	1	0	1	2	0.8675	998	737	265				
4732	2	0	0	0	1	0	0	1	0	0	0	1	1	0.3685	737	658	605				
13432	2	0	0	0	0	0	1	0	0	0	1	1	1	0.522388	630	511	65				
21858	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	65						
15	1	0	0	0	0	1	0	0	0	0	0	0	-1	1	996	1004					
4747	1	0	0	0	0	0	0	1	0	0	0	0	-1	1	1395	605					
13445	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	1204						
9185	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	1862						
30	2	0	0	0	0	1	0	0	0	0	1	0	-1	1	995	737	268				
4762	2	0	0	0	1	0	0	1	0	0	0	0	-1	1	737	658	605				
9200	3	0	0	0	1	0	0	1	0	0	1	1	2	0.691398	658	628	511	63			
13459	2	0	0	0	0	0	1	0	0	0	1	1	1	0.522463	628	511	63				
21871	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	63						
45	2	0	0	0	0	1	0	0	0	0	1	0	-1	1	994	737	269				
4777	2	0	0	0	1	0	0	1	0	0	0	1	2	0.6965	737	656	607				
9215	3	0	0	0	1	0	0	1	0	0	1	2	1	0.353258	656	629	511	61			
17942	1	0	0	0	0	0	0	0	0	1	0	1	1	0.893357	511	61					
60	2	0	0	0	0	1	0	0	0	1	0	2	1	0.4965	993	738	269				
13487	2	0	0	0	0	0	1	0	0	0	1	0	-1	1	628	510	61				
17957	1	0	0	0	0	0	0	0	0	1	0	0	-1	1	510	61					
21901	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	61						
75	2	0	0	0	0	1	0	0	0	1	0	0	-1	1	991	738	271				
4807	2	0	0	0	1	0	0	1	0	0	0	0	-1	1	738	657	605				
9243	3	0	0	0	1	0	0	1	0	0	1	2	2	0.693305	657	627	510	58			
13502	2	0	0	0	0	0	1	0	0	0	1	2	1	0.524686	627	510	58				
90	2	0	0	0	0	1	0	0	0	1	0	1	1	0.4955	991	737	272				
9258	3	0	0	0	1	0	0	1	0	0	1	0	-1	1	656	629	510	55			

First Sheet | Second Sheet | Third Sheet | Forth Sheet | Fifth Sheet

Normal View | Ready | Sum=0

Figure A.1 Examples of the results generated by the *VMC*

With regards to co-residency, the next section describes the *PA*s that are implemented by the *VMC* simulator.

A.2 Implemented VM Placement Algorithms

Given a pool of hosts in the IaaS cloud (host 1, host 2, ..., host ω) that are distributed in different clusters (cluster 1, cluster 2, …, cluster C) and a sequence of VMs requests, the *PA*s specify in which cluster and host a newly created VM should be placed. In case all hosts are full, no placement takes place. The system model considers four *PA*s that are used in popular

IaaS cloud platforms including Eucalyptus [6], OpenNebula [60], Nimbus [85] and OpenStack [98]. Based on the previous system model assumptions, the *VMC* simulator implements the following *PA*s.

A.2.1 First Fit

First Fit places a new VM as follows:

- Placement in clusters level:
  - If all clusters are *full*, then the new VM cannot be placed. Else
  - Select the lowest available cluster *i* then go to Placement in hosts level.
- Placement in hosts level:
  - Place the new VM in the lowest available host *j* from cluster *i*.

A.2.2 Next Fit

Next Fit mainly focuses on distributing the VMs equally between clusters and hosts with the help of the following pointers:

- pointer_i : Initially points to the lowest cluster (i.e. i =1).
- pointer_j for each cluster i: Initially points to the lowest host that belongs to cluster *i*.

Next fit places a new VM to hosts in a cyclic manner as follows:

- Placement in clusters level:

1. If all clusters are *full*, then the new VM cannot be placed. Else
2. If the cluster indicated by pointer_i is *full*:
   a. Move pointer_i to point to the next cluster i+1, given that i+1≤ C. Otherwise move it to point to the lowest cluster (i=1).
   b. Repeat the Placement in clusters level.
3. If the cluster indicated by pointer_i is *available*:
   a. Select cluster *i* for placement.
   b. Move pointer_i to point to the next cluster i+1, given that i+1≤ C. Otherwise move it to point to the lowest cluster (i=1).
   c. Go to Placement in hosts level.

- Placement in hosts level:

Now that an available cluster *i* is selected:

1. If the host indicated by pointer_j is *full*:
    a. Move pointer_i to point to the next host $j+1$, given that $j+1 \leq \omega$. Otherwise move it to point to the lowest host ($j=1$).
    b. Repeat the Placement in hosts level.
2. If the host indicated by pointer_j is *available*:
    a. Place the new VM in host *j*.
    b. Move pointer_j to point to the next host $j+1$, given that $j+1 \leq \omega$. Otherwise move it to point to the lowest host ($j=1$).

A.2.3 Power Save

Power Save is similar to First Fit but with a number of differences:

- Power Save puts a host to *sleep* mode when the host contains zero VM. A *sleep* mode indicates that the host is unavailable for placing new VMs.
- Power Save reawakens a host from sleep mode to place new VMs when all the other non-sleeping hosts are full.
- Initially: Power Save puts all hosts into *sleep* mode except host 1 in cluster 1.
- Upon receiving a VM placement request: Whenever a VM request is received, Power Save checks non-sleeping hosts in all clusters if all of them are *full* then:
    - Select the lowest cluster that has a sleeping host,
    - Awaken the lowest sleeping host.

Power Save places a new VM as follows:

- Placement in clusters level:
    1. If all clusters are *full*, then the new VM cannot be placed. Else
    2. Select the lowest available cluster *i* then go to Placement in hosts level.
- Placement in hosts level:
    1. Place the new VM in the lowest available host *j* from cluster *i*.

A.2.4 Random

Random places new VMs in a rather straightforward way compared to the other algorithms:

- Placement in clusters level:
  1. If all clusters are *full*, then the new VM cannot be placed. Else
  2. Select a cluster *i* uniformly at random. If it is *full,* then keep selecting random clusters until an *available* cluster *i* is found, then go to Placement in hosts level.

- Placement in hosts level:
  1. From the selected cluster *i*, select a host j uniformly at random. If it is *full* then keep selecting random hosts until an *available* host *j* is found
  2. Place the new VM in *j*.

Appendix B

Designing a Fractional Factorial Experiment

B.1 Fractional Factorial Definition

Fractional factorial design is an effective experimental approach that is widely used in industrial experiments [33]. When there are too many parameters and levels to be included in a limited resources experiment, fractional factorial helps to construct a reduced and balanced experiment design. Fractional factorial experiments are used in Chapter 4 to identify the top influential parameters and interactions on the co-residency metrics in an effective way. The basic concept of fractional factorial design is to include a subset (fraction) of the experimental runs that only cover important parameter combinations and interactions. This is in contrast to the full factorial experimental approach which includes all parameter combinations. A 2-way fractional factorial design of an IV resolution is used in Chapter 4 that ensures that the effect of a given parameter does not confound with the effects of any other parameter and 2-parameter interactions.

Fractional factorial designs are expressed in this thesis using the following notation:

$$\boldsymbol{L_{\mathrm{r}}}^{\boldsymbol{p-s}}$$

Where:

- L is the number of levels used to examine each parameter (i.e. L is always 2 in Chapter 4),
- r is the design resolution which specifies the degree to which the effect of each parameter confounds with the other parameters and interactions (i.e. r is chosen to be of resolution IV),
- p is the number of parameters under investigation (i.e. eight parameters in Chapter 4), and
- s represents the size of the fraction that is selected from the original full factorial design.

B.2 Designing a 2_{IV}^{4} Fractional Factorial Experiment

The aim in this section is to design a fractional factorial experiment to identify the most influential parameters and interactions on the co-residency metrics in Chapter 4. Using the eight parameters with two levels from Table 4.4, a full factorial experimental design that covers all possible parameter combinations will result in 2^8 experimental runs. In order to reduce the experiment size, the following steps are applied to design a 2_{IV}^{8-4} fractional factorial experiment that uses a $\frac{1}{2^4}$ fraction of the 2^8 experimental runs in the full factorial design:

1. Starting with X1, X2, X3 and X4 as the design parameters, construct a full factorial design of *p*-*s* parameters (i.e. 8-4 = 4) that has 2^4 experimental runs. These runs cover all possible parameter combinations (Table B.1).

Run	X1	X2	X3	X4
1	Low	Low	Low	Low
2	Low	High	High	High
3	High	Low	High	Low
4	High	High	Low	Low
5	High	Low	Low	Low
6	High	Low	High	High
7	Low	High	Low	High
8	High	High	Low	High
9	Low	Low	High	Low
10	Low	High	High	Low
11	High	High	High	Low
12	Low	High	Low	Low
13	High	Low	Low	High
14	Low	Low	High	High
15	Low	Low	Low	High
16	High	High	High	High

Table B.1 Constructing a full factorial experiment using 4 parameters

2. Then all the possible interactions between these four parameters are added in new columns (Table B.2). The new columns are simply the multiplication between the interaction parameters levels. Multiplying the same level results in a High level, and multiplying different level results in a Low level:

Run	X1	X2	X3	X4	X1X2	X1X3	X1X4	X2X3	X2X4	X3X4	X1X2X3	X1X2X4	X1X3X4	X2X3X4	X1X2X3X4
1	Low	Low	Low	Low	High	High	High	High	High	High	Low	Low	Low	Low	High
2	High	Low	Low	Low	Low	Low	Low	High	High	High	High	High	High	Low	Low
3	Low	High	Low	Low	Low	High	High	Low	Low	High	High	High	Low	High	Low
4	High	High	Low	Low	High	Low	Low	Low	Low	High	Low	Low	High	High	High
5	Low	Low	High	Low	High	Low	High	Low	High	Low	High	Low	High	High	Low
6	High	Low	High	Low	Low	High	Low	Low	High	Low	Low	High	Low	High	High
7	Low	High	High	Low	Low	Low	High	High	Low	Low	Low	High	High	Low	High
8	High	High	High	Low	High	High	Low	High	Low	Low	High	Low	Low	Low	Low
9	Low	Low	Low	High	High	High	Low	High	Low	Low	Low	High	High	High	Low
10	High	Low	Low	High	Low	Low	High	High	Low	Low	High	Low	Low	High	High
11	Low	High	Low	High	Low	High	Low	Low	High	Low	High	Low	High	Low	High
12	High	High	Low	High	High	Low	High	Low	High	Low	Low	High	Low	Low	Low
13	Low	Low	High	High	High	Low	Low	Low	Low	High	High	High	Low	Low	High
14	High	Low	High	High	Low	High	High	Low	Low	High	Low	Low	High	Low	Low
15	Low	High	High	High	Low	Low	Low	High	High	High	Low	Low	Low	High	Low
16	High	High	High	High	High	High	High	High	High	High	High	High	High	High	High

Table B.2 Adding all the possible interactions between the 4 parameters

3. The remaining parameters (X5, X6, X7 and X8) are carefully substituted with redundant high-order interactions of the first 4 parameters (i.e. 3-parameter interactions). There are a number of standard approaches to substitute redundant interactions with parameters in a resolution IV fractional factorial design suggested in [15]. Resolution IV fractional factorial designs ensure that a parameter's effect confounds with at worst 3-parameter interactions. Therefore, 3-parameter and higher interactions effects are not considered in identifying the influential parameters on the co-residency metrics in Chapter 4. The basic rule to choose which 3-parameter interactions are to be replaced with which parameter is that the effects of the

substituted 3-parameter interactions do not confound with the effects of the parameters and 2-parameter interactions. Table B.3 shows that the X5, X6, X7 and X8 parameters were substituted with the following 3-parameter interactions:

X5 = X2X3X4 X6 = X1X3X4 X7 = X1X2X3 X8 = X1X2X4

X7 X8 X6 X5

Run	X1	X2	X3	X4	X1X2	X1X3	X1X4	X2X3	X2X4	X3X4	X1X2X3	X1X2X4	X1X3X4	X2X3X4	X1X2X3X4
1	Low	Low	Low	Low	High	High	High	High	High	High	Low	Low	Low	Low	High
2	High	Low	Low	Low	Low	Low	Low	High	High	High	High	High	High	Low	Low
3	Low	High	Low	Low	Low	High	High	Low	Low	High	High	High	Low	High	Low
4	High	High	Low	Low	High	Low	Low	Low	Low	High	Low	Low	High	High	High
5	Low	Low	High	Low	High	Low	High	Low	High	Low	High	Low	High	High	Low
6	High	Low	High	Low	Low	High	Low	Low	High	Low	Low	High	Low	High	High
7	Low	High	High	Low	Low	Low	High	High	Low	Low	Low	High	High	Low	High
8	High	High	High	Low	High	High	Low	High	Low	Low	High	Low	Low	Low	Low
9	Low	Low	Low	High	High	High	Low	High	Low	Low	Low	High	High	High	Low
10	High	Low	Low	High	Low	Low	High	High	Low	Low	High	Low	Low	High	High
11	Low	High	Low	High	Low	High	Low	Low	High	Low	High	Low	High	Low	High
12	High	High	Low	High	High	Low	High	Low	High	Low	Low	High	Low	Low	Low
13	Low	Low	High	High	High	Low	Low	Low	Low	High	High	High	Low	Low	High
14	High	Low	High	High	Low	High	High	Low	Low	High	Low	Low	High	Low	Low
15	Low	High	High	High	Low	Low	Low	High	High	High	Low	Low	Low	High	Low
16	High	High	High	High	High	High	High	High	High	High	High	High	High	High	High

Table B.3 Replacing X5, X6, X7 and X8 parameters with 3-parameter interactions.

4. The final 2_{IV}^{4} fractional factorial experiment consists of 16 experimental runs covering eight parameters combinations in a balanced fashion (Table B.4). Each parameter is tested at each of its two levels in eight runs in order to increase the validity of the results.

Run	X1	X2	X3	X4	X5	X6	X7	X8
1	Low	Low	Low	Low	Low	Low	Low	Low
2	High	Low	Low	Low	Low	High	High	High
3	Low	High	Low	Low	High	Low	High	High
4	High	High	Low	Low	High	High	Low	Low
5	Low	Low	High	Low	High	High	High	Low
6	High	Low	High	Low	High	Low	Low	High
7	Low	High	High	Low	Low	High	Low	High
8	High	High	High	Low	Low	Low	High	Low
9	Low	Low	Low	High	High	High	Low	High
10	High	Low	Low	High	High	Low	High	Low
11	Low	High	Low	High	Low	High	High	Low
12	High	High	Low	High	Low	Low	Low	High
13	Low	Low	High	High	Low	Low	High	High
14	High	Low	High	High	Low	High	Low	Low
15	Low	High	High	High	High	Low	Low	Low
16	High	High	High	High	High	High	High	High

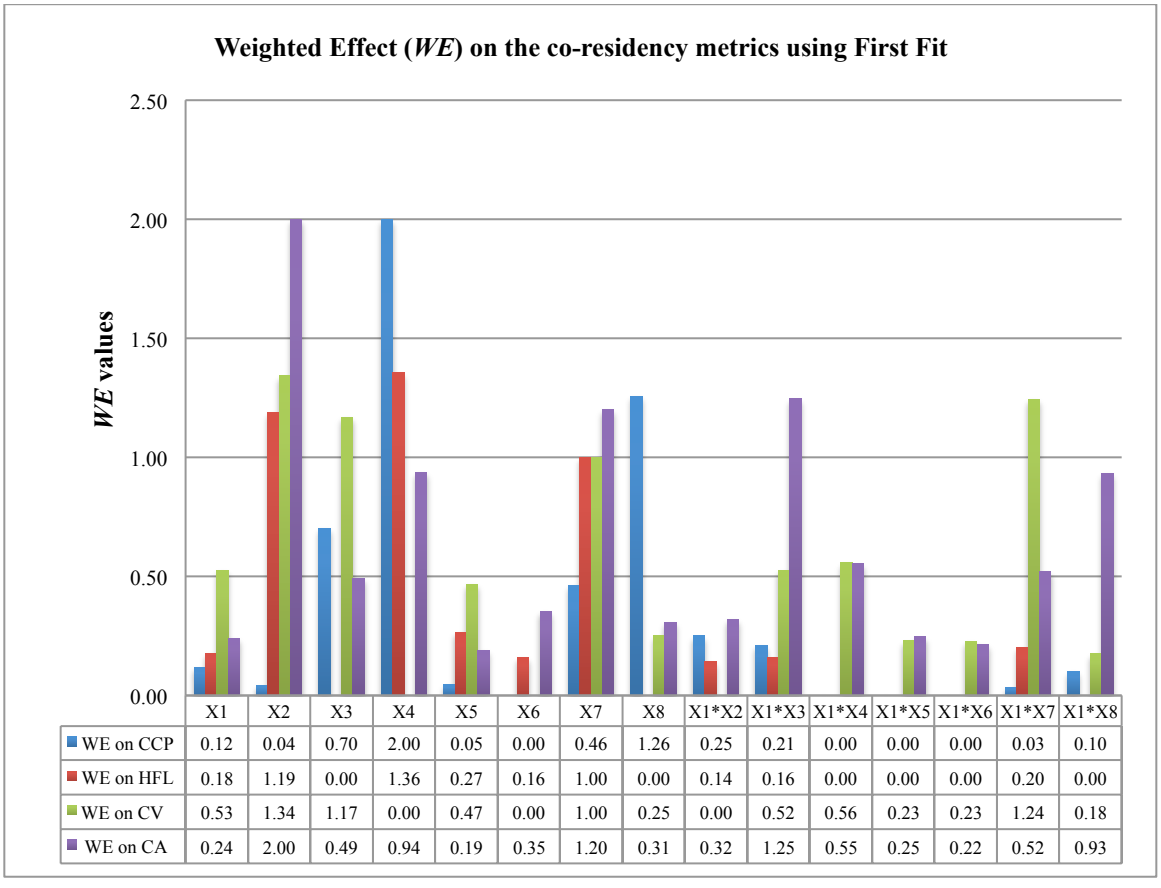
Table B.4 Final design of the 2_{IV}^{8-4} fractional factorial experiment

Appendix C

Weighted Effects on the Co-residency Metrics

The results of the parameters and 2-parameter interactions Weighted Effects *WEs* on each of the co-residency metrics are provided in this Appendix. These *WEs* are calculated under First Fit, Next Fit, Power Save and Random. The following figures show the parameters and 2-parameter interactions in the x-axis and the corresponding *WEs* in the y-axis. As per the definition of *WE* in Section 4.3.4.3, the maximum *WE* on a given co-residency metric that parameters and 2-parameter interactions can achieve is two.

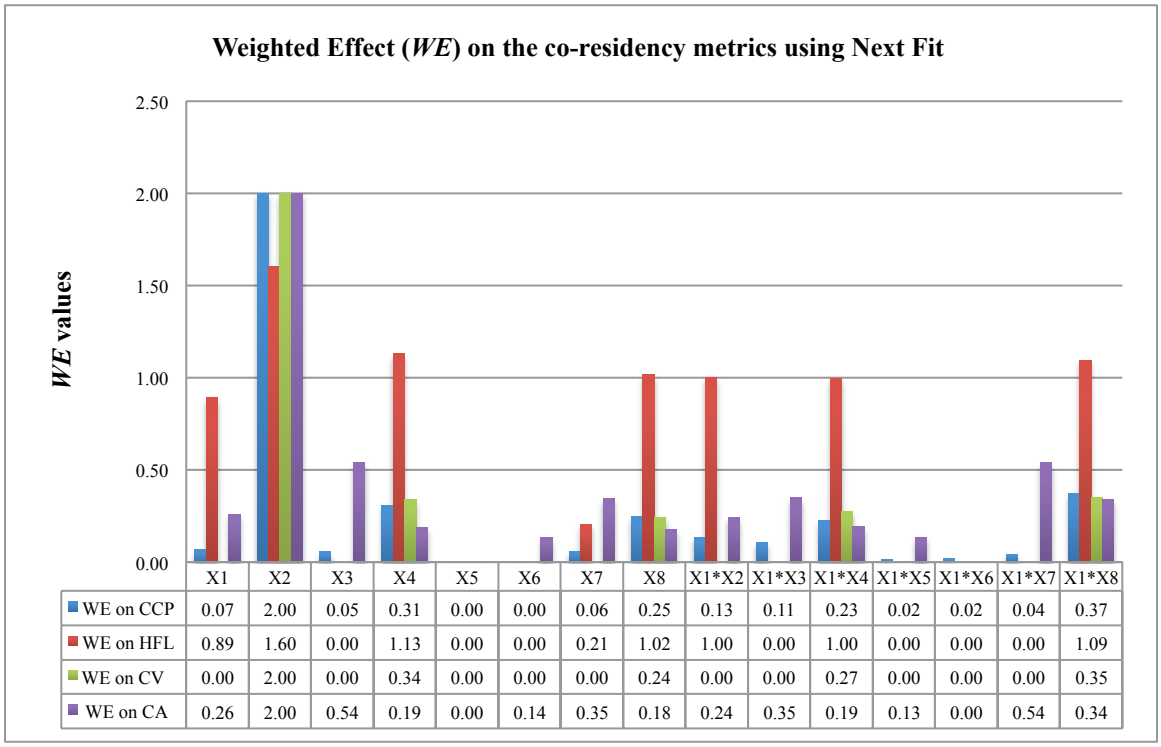
When using First Fit as the *PA* (Figure C.1), the results showed that User Arrival Rate (X4) achieved the highest *WE* on the *CCP* metric followed by the VMs Request Rate (X8). In addition, Users' Arrival Rate (X4) achieved the highest *WE* on the *HFL* metric followed by Number of Hosts (X2). However, Number of Hosts (X2) repeatedly scored the highest WE on the *CV* and *CA* metrics.



	X1	X2	X3	X4	X5	X6	X7	X8	X1*X2	X1*X3	X1*X4	X1*X5	X1*X6	X1*X7	X1*X8
WE on CCP	0.12	0.04	0.70	2.00	0.05	0.00	0.46	1.26	0.25	0.21	0.00	0.00	0.00	0.03	0.10
WE on HFL	0.18	1.19	0.00	1.36	0.27	0.16	1.00	0.00	0.14	0.16	0.00	0.00	0.00	0.20	0.00
WE on CV	0.53	1.34	1.17	0.00	0.47	0.00	1.00	0.25	0.00	0.52	0.56	0.23	0.23	1.24	0.18
WE on CA	0.24	2.00	0.49	0.94	0.19	0.35	1.20	0.31	0.32	1.25	0.55	0.25	0.22	0.52	0.93

Figure C.1 Weighted Effect *WE* on the co-residency metrics using First Fit

The *WEs* of the parameters and interactions when using Next Fit were different from the *WEs* when First Fit was used (Figure C.2). The results showed that Number of Hosts (X2) scored the highest *WE* on the *CCP* metric. However, the rest of the parameters and interactions scored relatively smaller *WEs*. In addition, Number of Hosts (X2), User Arrival Rate (X4) and VMs per Request (X8) scored the highest *WEs* on the *HFL* metric. Again, Number of Hosts (X2) repeatedly scored the maximum *WE* on the *CV and CA* metrics. In general, 2-parameter interactions scored more *WEs* on the metrics when using Next Fit compared to First Fit.

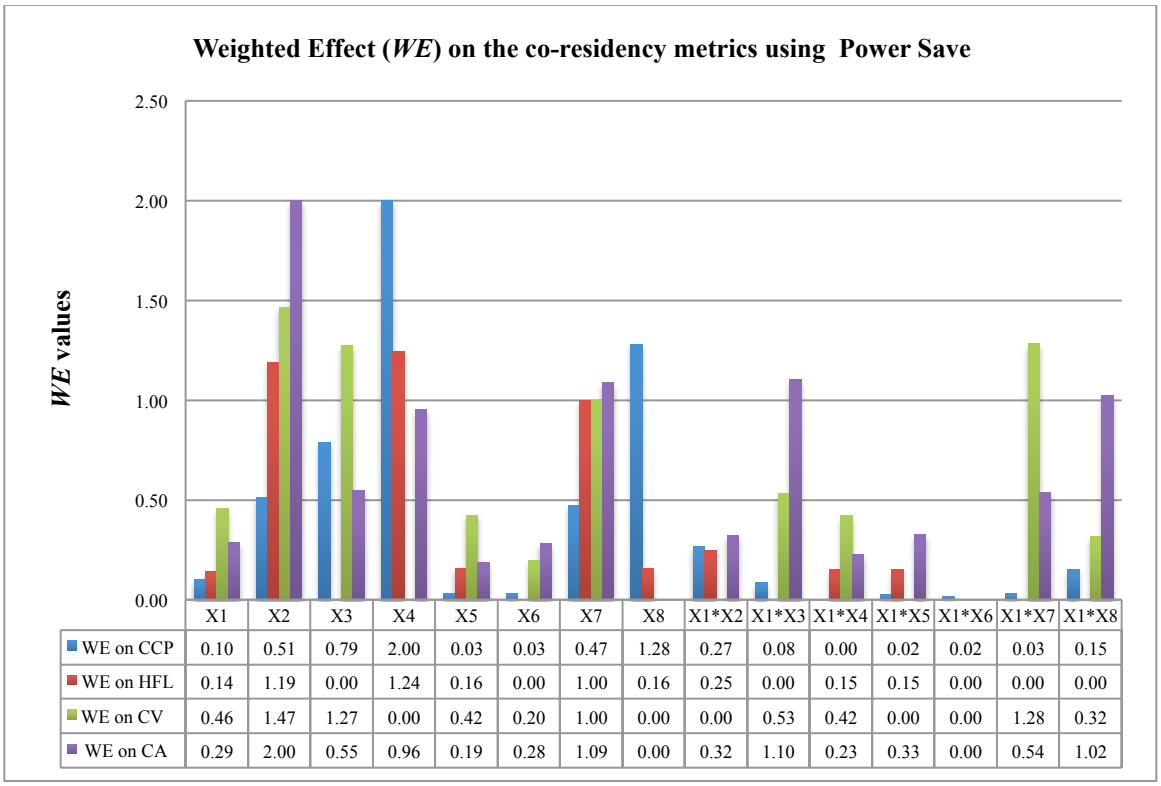


	X1	X2	X3	X4	X5	X6	X7	X8	X1*X2	X1*X3	X1*X4	X1*X5	X1*X6	X1*X7	X1*X8
WE on CCP	0.07	2.00	0.05	0.31	0.00	0.00	0.06	0.25	0.13	0.11	0.23	0.02	0.02	0.04	0.37
WE on HFL	0.89	1.60	0.00	1.13	0.00	0.00	0.21	1.02	1.00	0.00	1.00	0.00	0.00	0.00	1.09
WE on CV	0.00	2.00	0.00	0.34	0.00	0.00	0.00	0.24	0.00	0.00	0.27	0.00	0.00	0.00	0.35
WE on CA	0.26	2.00	0.54	0.19	0.00	0.14	0.35	0.18	0.24	0.35	0.19	0.13	0.00	0.54	0.34

Figure C.2 Weighted Effect *WE* on the co-residency metrics using Next Fit

When using Power Save as the *PA* (Figure C.3), Users' Arrival Rate (X4) achieved the highest *WE* on the *CCP* metric followed by the VMs per Request (X8). These two parameters scored the highest *WEs* on the *CCP* metric when using the First Fit. In addition, Users' Arrival Rate (X4) and Number of Hosts (X2) scored approximately similar *WEs* on the *HFL* metric. Again, these two parameters scored the highest *WEs* on the *HFL* metric under First Fit. Moreover, Number of Hosts (X2) scored the highest *WE* on the *CV* metric along with the interaction of Number of Clusters (X1) and VMs Average Lifetime (X7). Finally, Number of

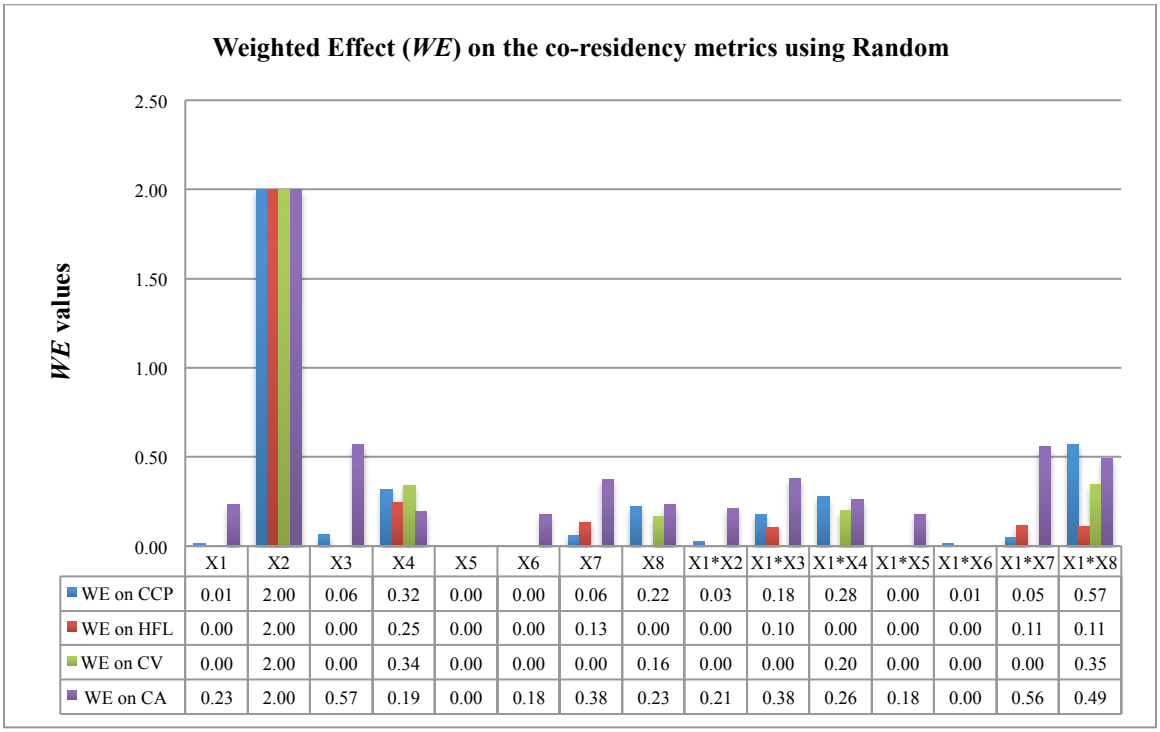
Hosts (X2) repeatedly scored the maximum *WE* on the *CA* metric. Further, VM Average Lifetime (X7) is an important driving parameter of *CA*.



	X1	X2	X3	X4	X5	X6	X7	X8	X1*X2	X1*X3	X1*X4	X1*X5	X1*X6	X1*X7	X1*X8
WE on CCP	0.10	0.51	0.79	2.00	0.03	0.03	0.47	1.28	0.27	0.08	0.00	0.02	0.02	0.03	0.15
WE on HFL	0.14	1.19	0.00	1.24	0.16	0.00	1.00	0.16	0.25	0.00	0.15	0.15	0.00	0.00	0.00
WE on CV	0.46	1.47	1.27	0.00	0.42	0.20	1.00	0.00	0.00	0.53	0.42	0.00	0.00	1.28	0.32
WE on CA	0.29	2.00	0.55	0.96	0.19	0.28	1.09	0.00	0.32	1.10	0.23	0.33	0.00	0.54	1.02

Figure C.3 Weighted Effect *WE* on the co-residency metrics using Power Save

With regards to the *WEs* when using Random (Figure C.4), the results were similar to Next Fit, but not quite to the same extent. In general, Number of Hosts (X2) can be seen as a parameter with a strong influence on the metrics when Random is used as the *PA*. For example, Number of Hosts (X2) scored the maximum *WE* on all metrics.



	X1	X2	X3	X4	X5	X6	X7	X8	X1*X2	X1*X3	X1*X4	X1*X5	X1*X6	X1*X7	X1*X8
WE on CCP	0.01	2.00	0.06	0.32	0.00	0.00	0.06	0.22	0.03	0.18	0.28	0.00	0.01	0.05	0.57
WE on HFL	0.00	2.00	0.00	0.25	0.00	0.00	0.13	0.00	0.00	0.10	0.00	0.00	0.00	0.11	0.11
WE on CV	0.00	2.00	0.00	0.34	0.00	0.00	0.00	0.16	0.00	0.00	0.20	0.00	0.00	0.00	0.35
WE on CA	0.23	2.00	0.57	0.19	0.00	0.18	0.38	0.23	0.21	0.38	0.26	0.18	0.00	0.56	0.49

Figure C.4 Weighted Effect *WE* on the co-residency metrics using Random

Appendix D

Significant 2-Parameter Interactions on the Co-residency Metrics

Chapter 4 identified the X1*X4 and X1*X8 2-parameter interactions to have an influence on the co-residency metrics under Next Fit and Random (Table 4.9). Examining the interaction between parameters can significantly enhance the evaluation of their influence on the co-residency metrics. The presence of a significant interaction indicates that the effect of one parameter on the co-residency metrics is different at different levels of the other parameter. Based on the results of the broad and narrow experiments (Table 4.5 and Table 4.6), 22 significant effects (i.e. p-value <0.05) on the co-residency metrics were caused by the 2-parameter interactions (between X1*X4 and X1*X8). The following figures show the statistically significant interaction effects on the co-residency metrics (i.e. *CCP, HFL, CV* and *CA*) under Next Fit and Random. The figures reveal that 14 of these 2-parameter interactions (nearly 63.6%) were able to reduce the co-residency probability (as defined in Section 5.1) when both X4 and X8 were in low levels.

D.1 Significant 2-Parameter Interactions Using Next Fit (Broad-Experiment)

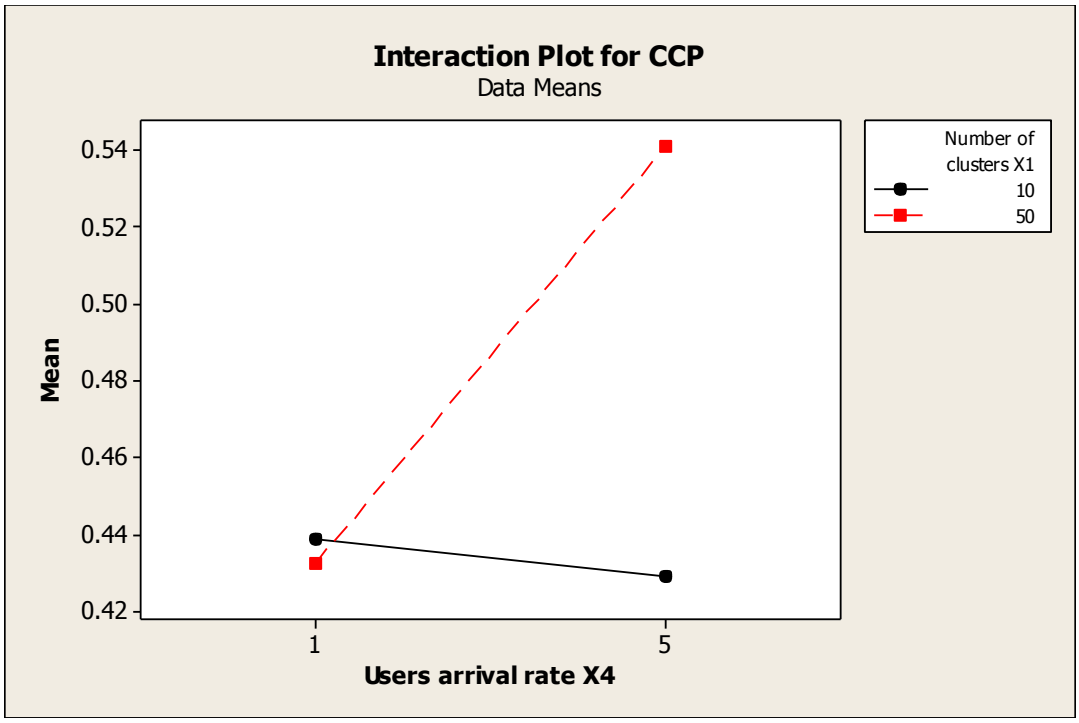


Figure D.1 Interaction plot for *CCP* between X1*X4

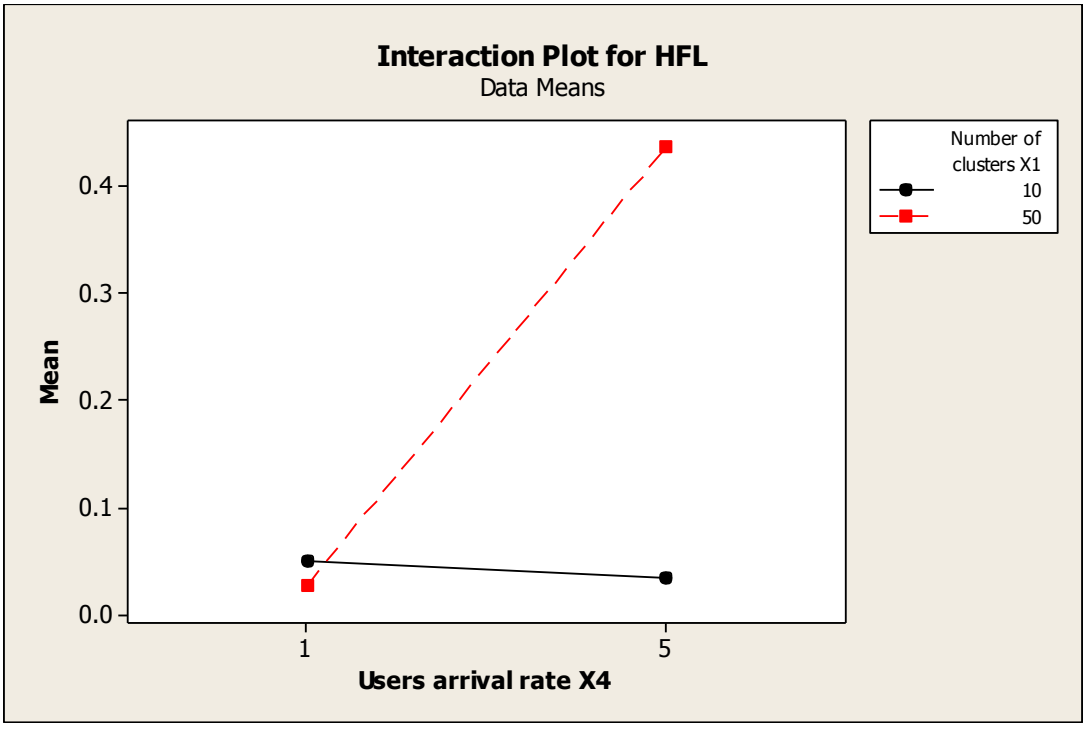


Figure D.2 Interaction plot for *HFL* between X1*X4

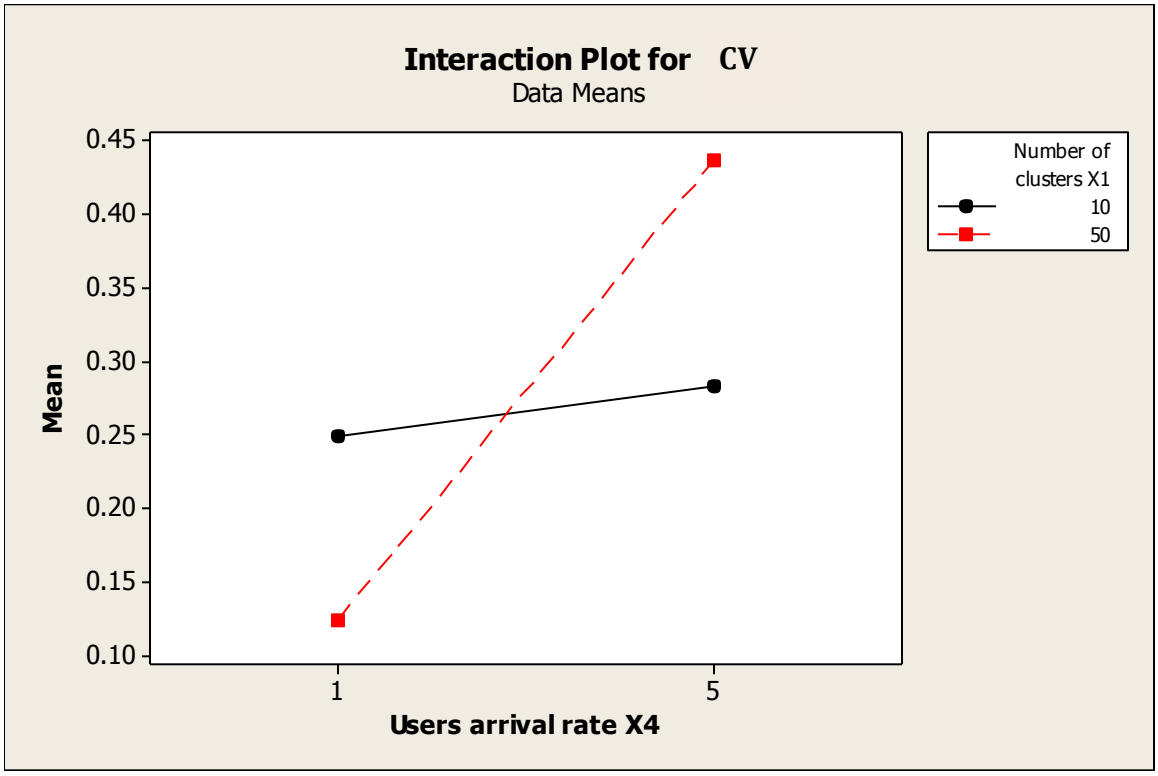


Figure D.3 Interaction plot for CV between X1*X4

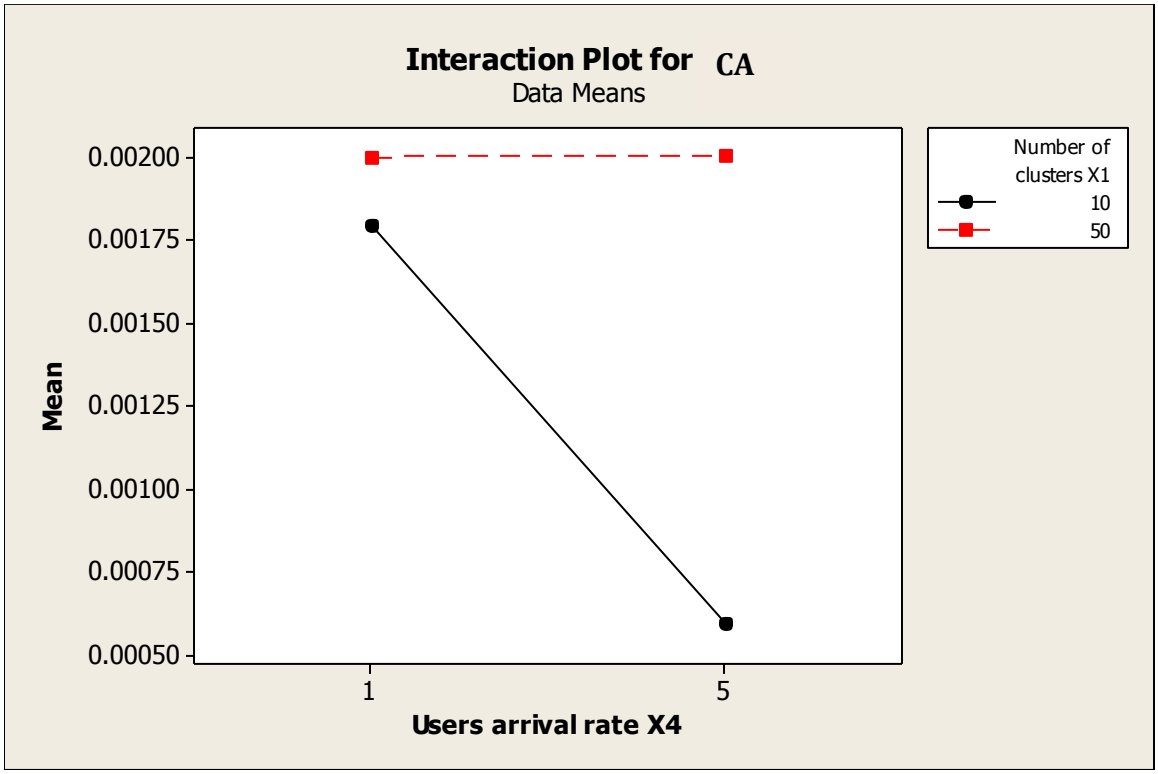


Figure D.4 Interaction plot for *CA* between X1*X4

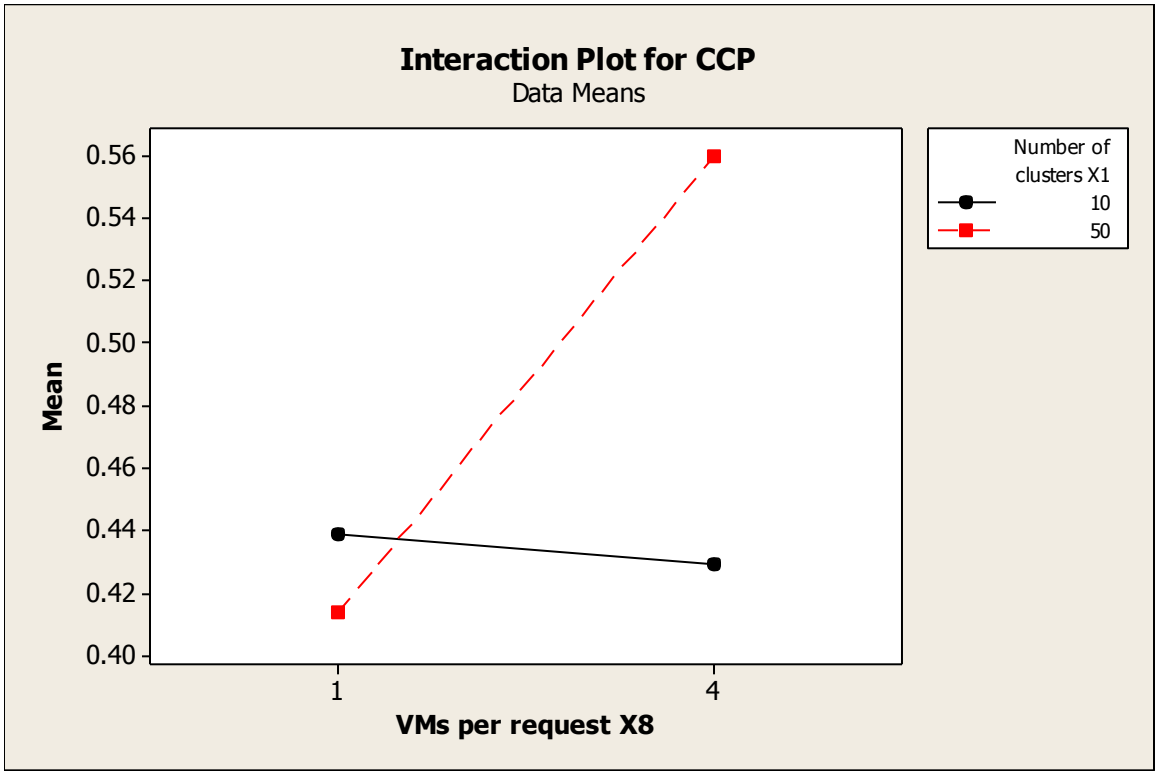


Figure D.5 Interaction plot for *CCP* between X1*X8

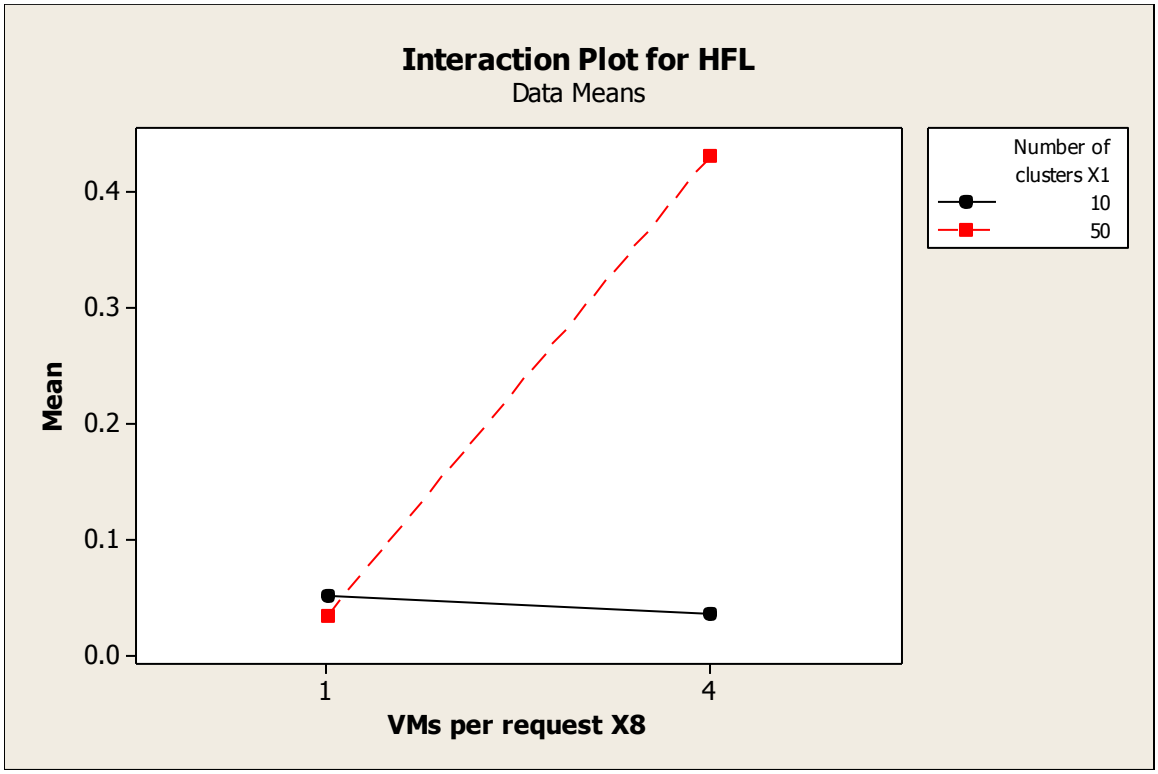


Figure D.6 Interaction plot for *HFL* between X1*X8

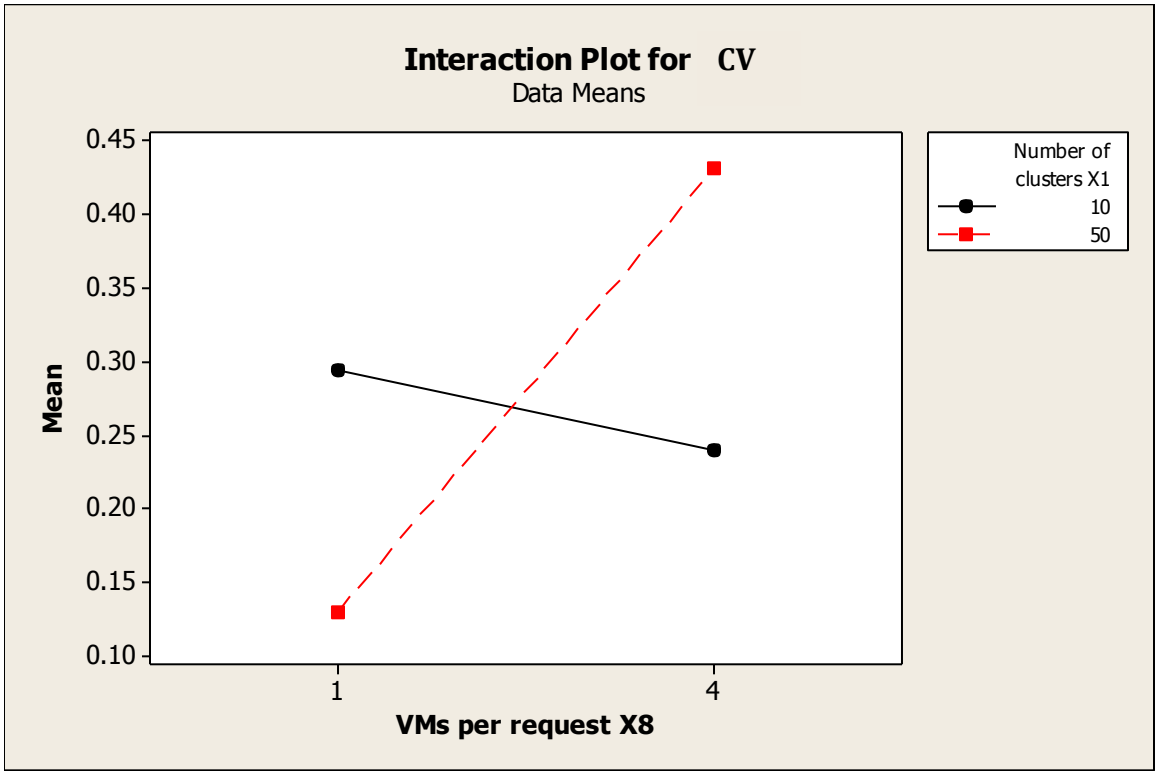


Figure D.7 Interaction plot for CV between X1*X8

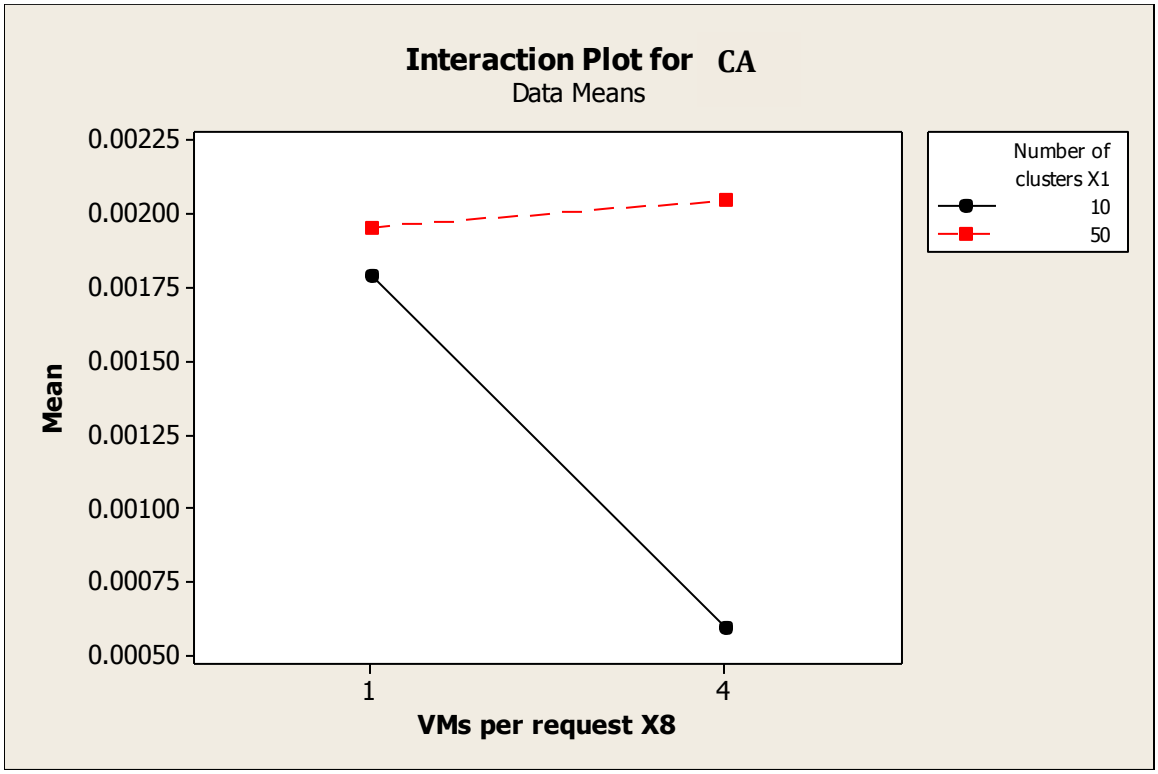


Figure D.8 Interaction plot for *CA* between X1*X8

D.2 Significant 2-Parameter Interactions Using Next Fit (Narrow-Experiment)

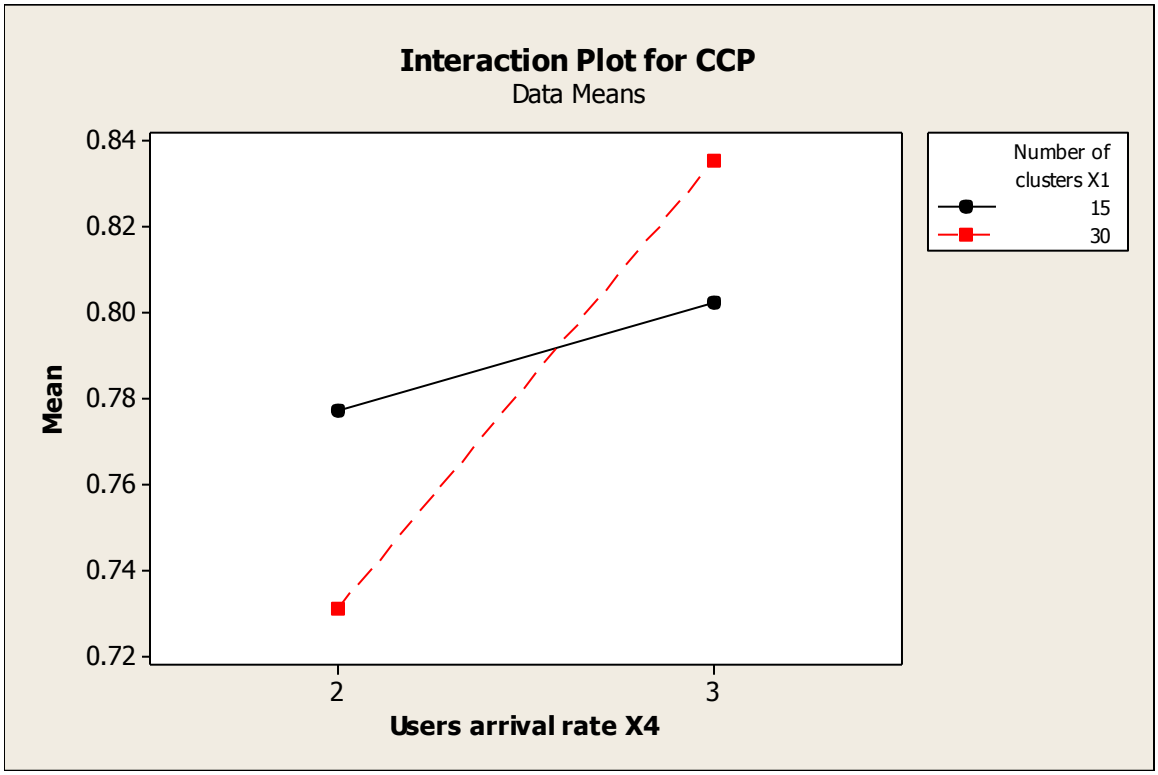


Figure D.9 Interaction plot for *CCP* between X1*X4

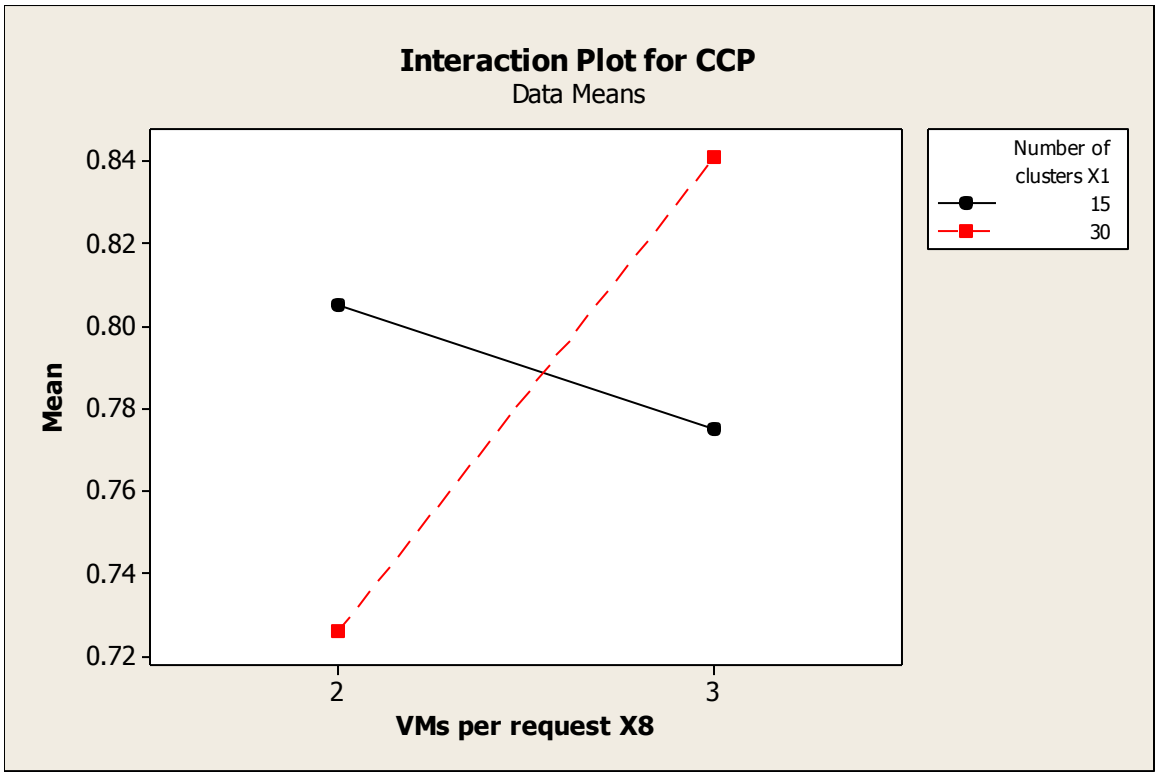


Figure D.10 Interaction plot for *CCP* between X1*X8

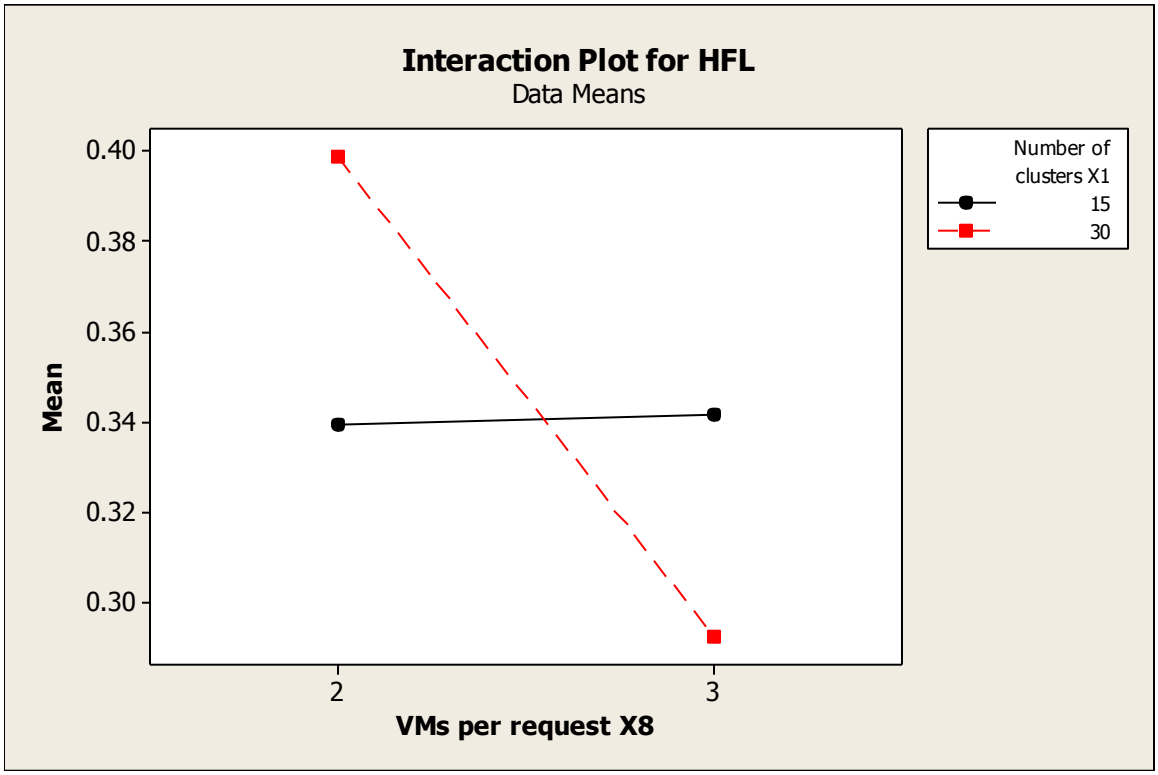


Figure D.11 Interaction plot for *HFL* between X1*X8

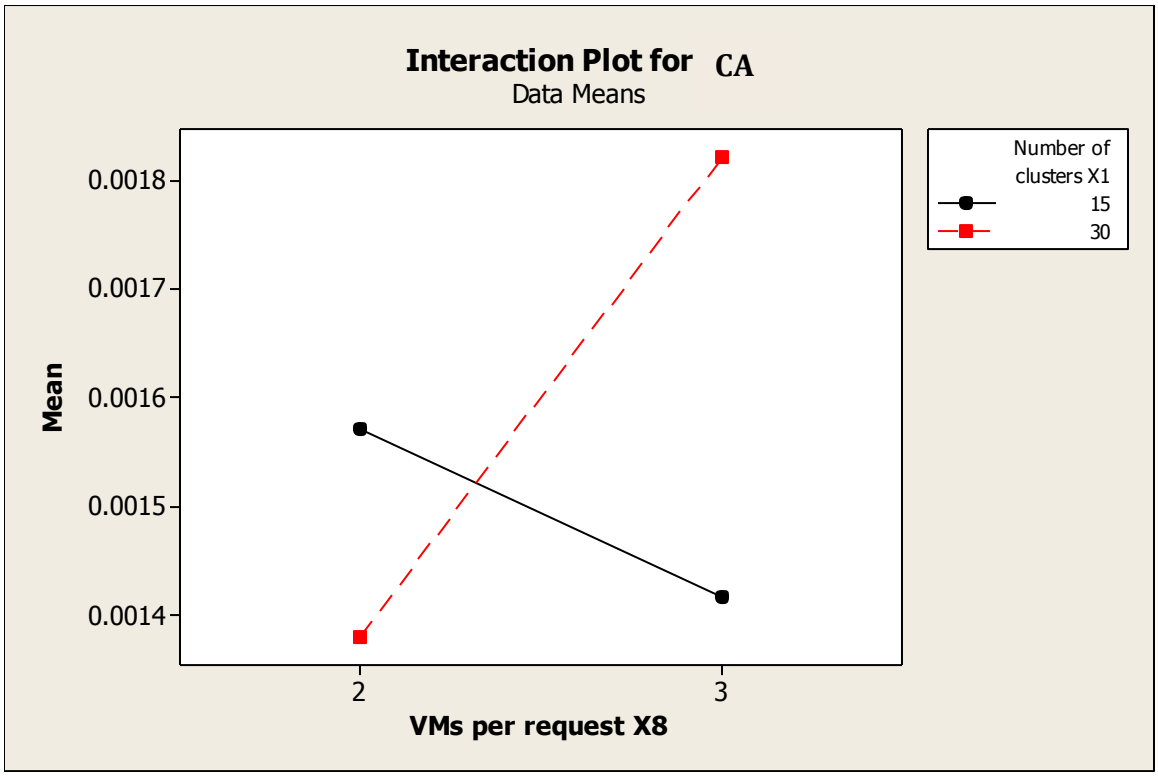


Figure D.12 Interaction plot for *CA* between X1*X8

D.3 Significant 2-Parameter Interactions Using Random (Broad-Experiment)

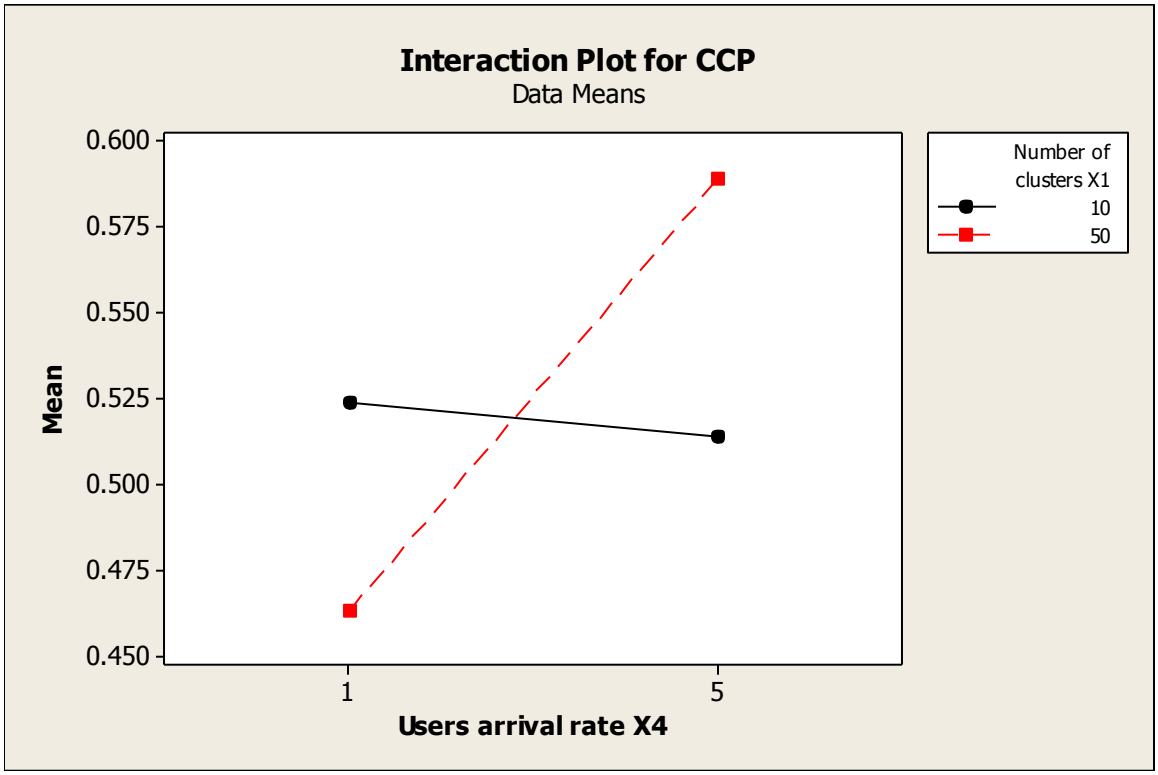


Figure D.13 Interaction plot for *CCP* between X1*X4

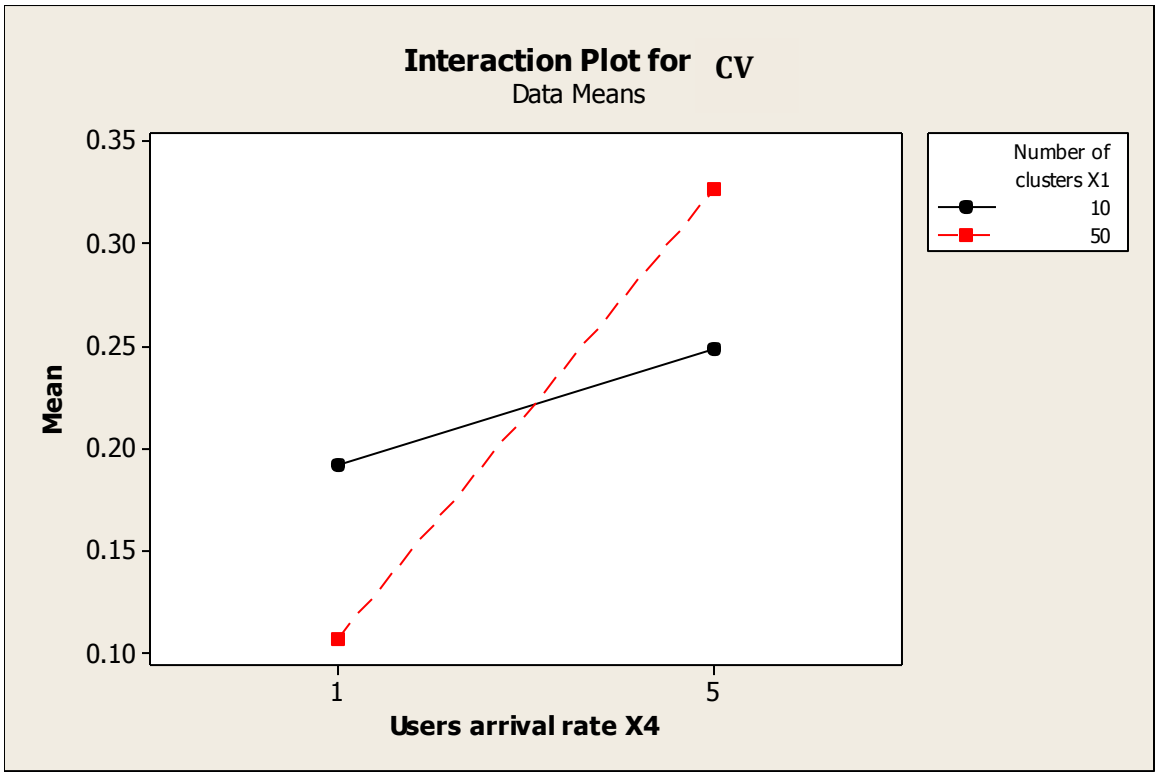


Figure D.14 Interaction plot for CV between X1*X4

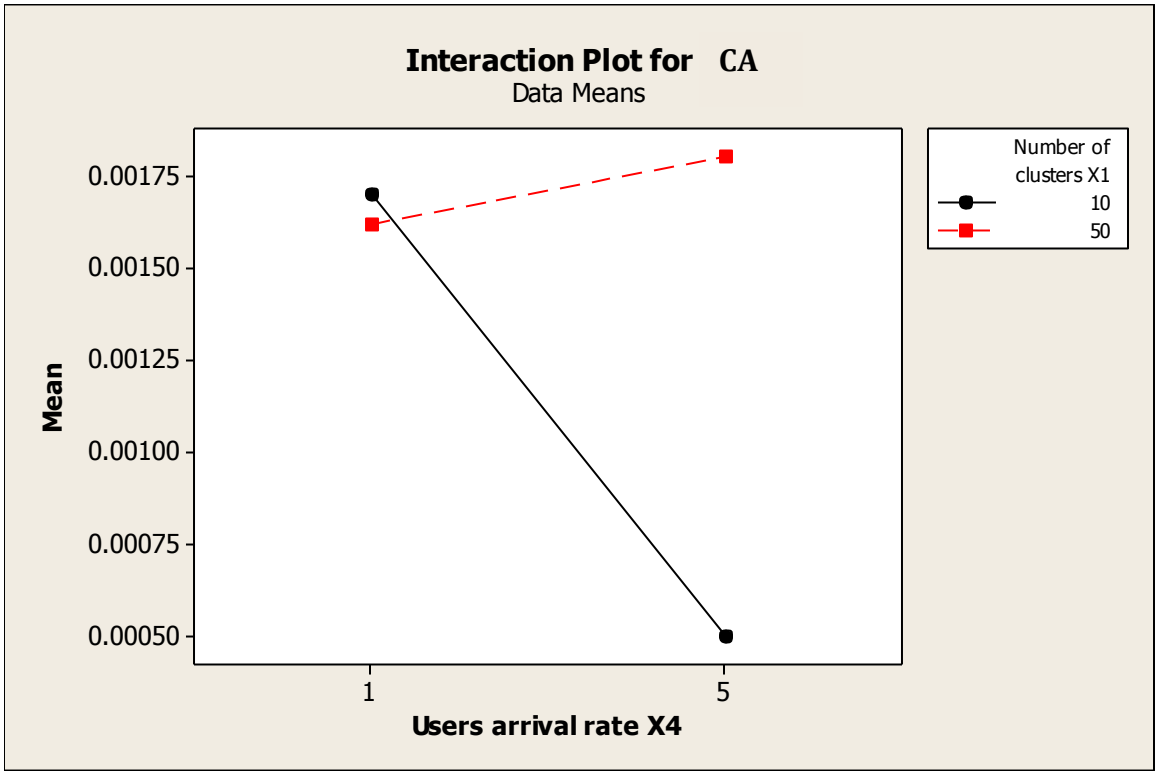


Figure D.15 Interaction plot for *CA* between X1*X4

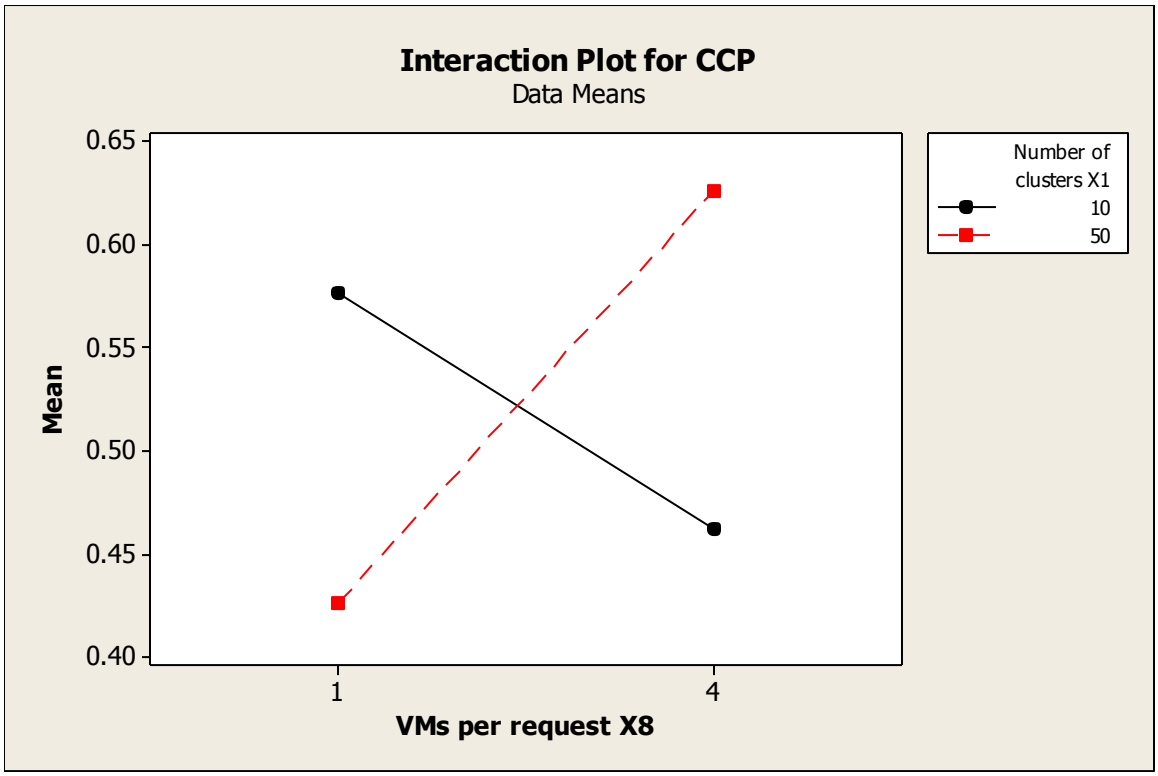


Figure D.16 Interaction plot for *CCP* between X1*X8

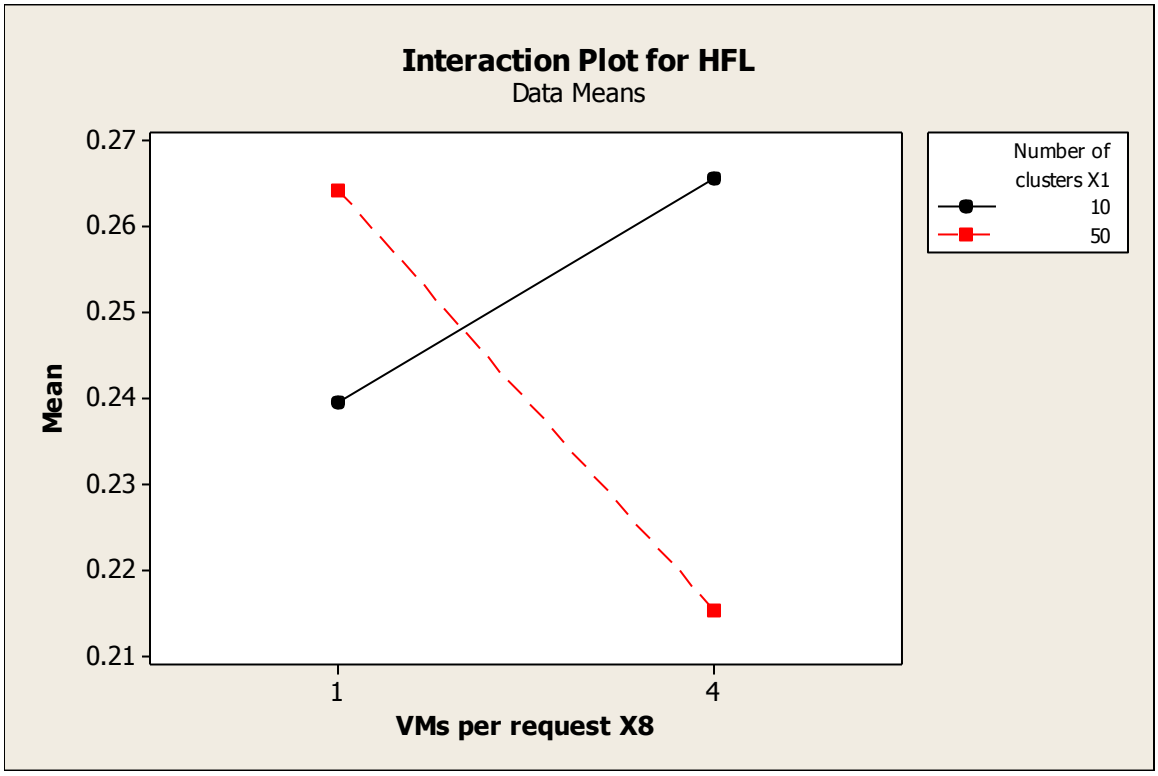


Figure D.17 Interaction plot for *HFL* between X1*X8

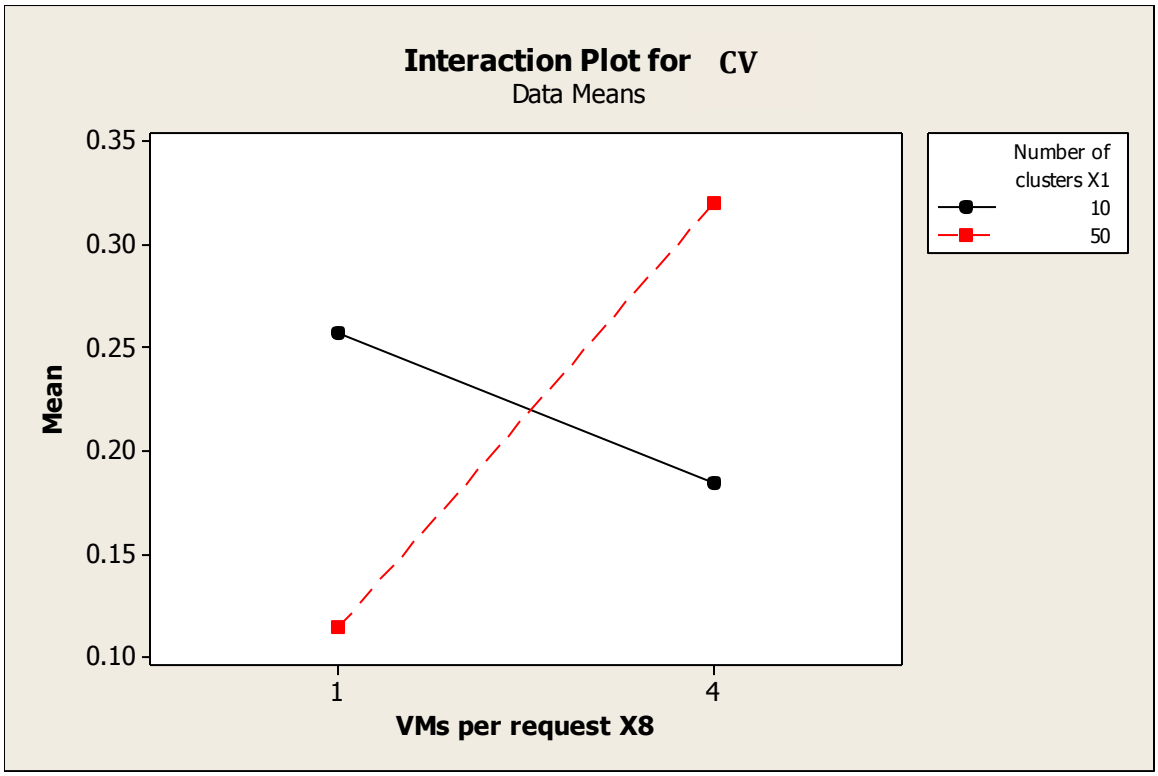


Figure D.18 Interaction plot for CV between X1*X8

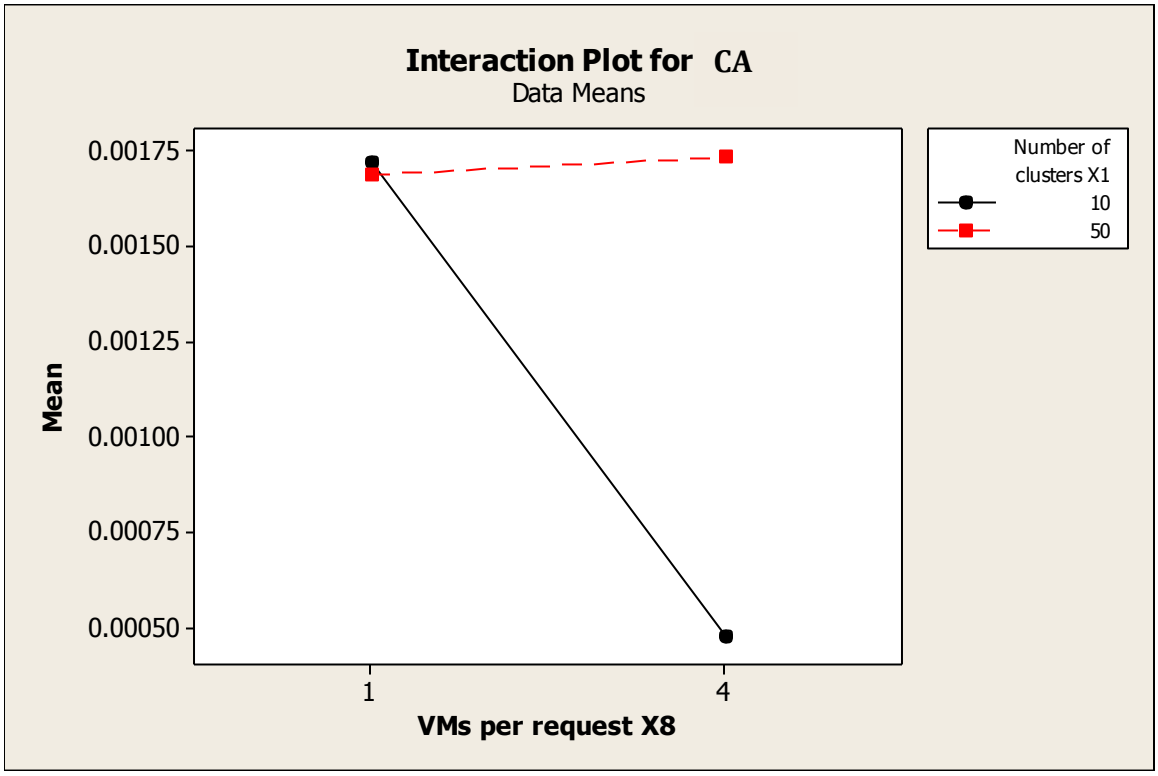


Figure D.19 Interaction plot for *CA* between X1*X8

D.4 Significant 2-Parameter Interactions Using Random (Narrow-Experiment)

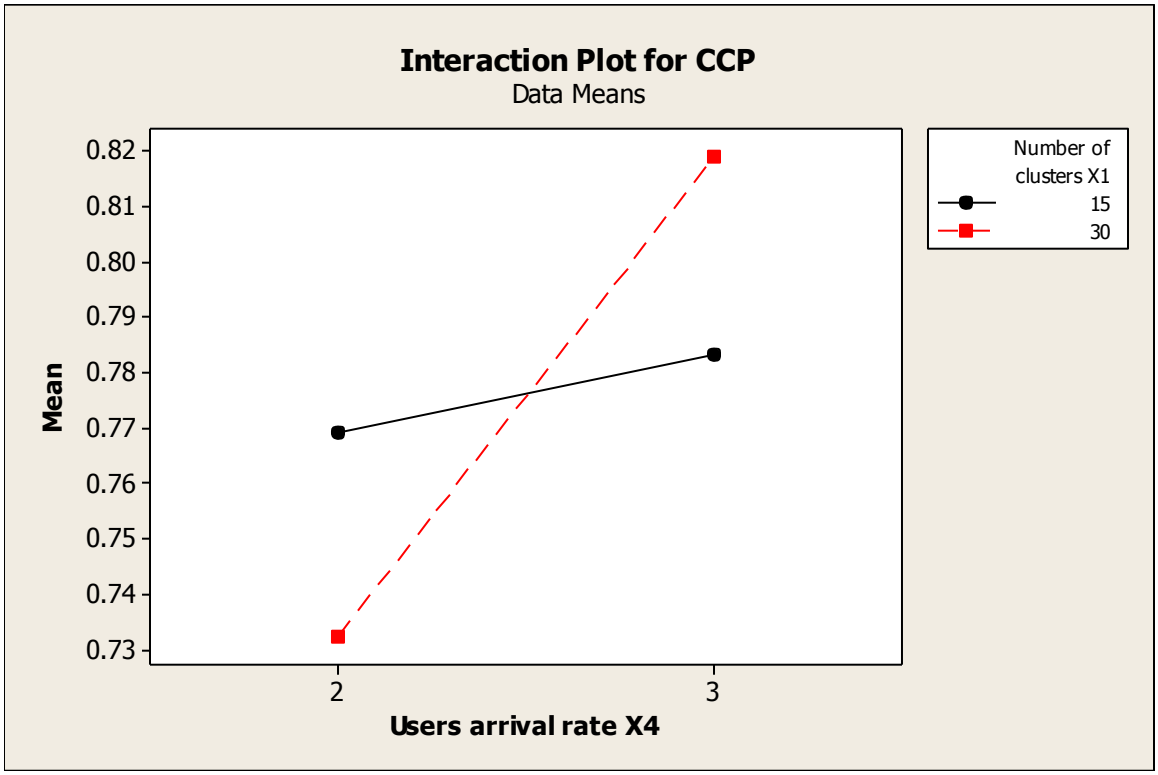


Figure D.20 Interaction plot for *CCP* between X1*X4

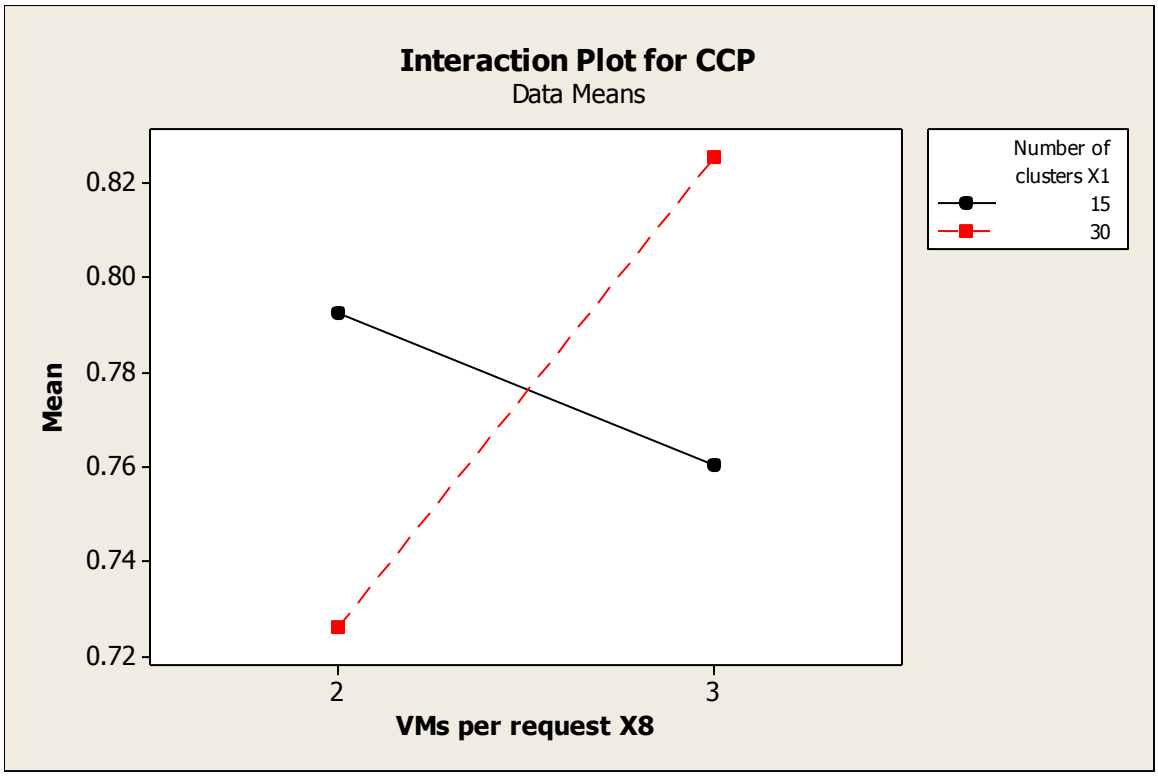


Figure D.21 Interaction plot for *CCP* between X1*X8

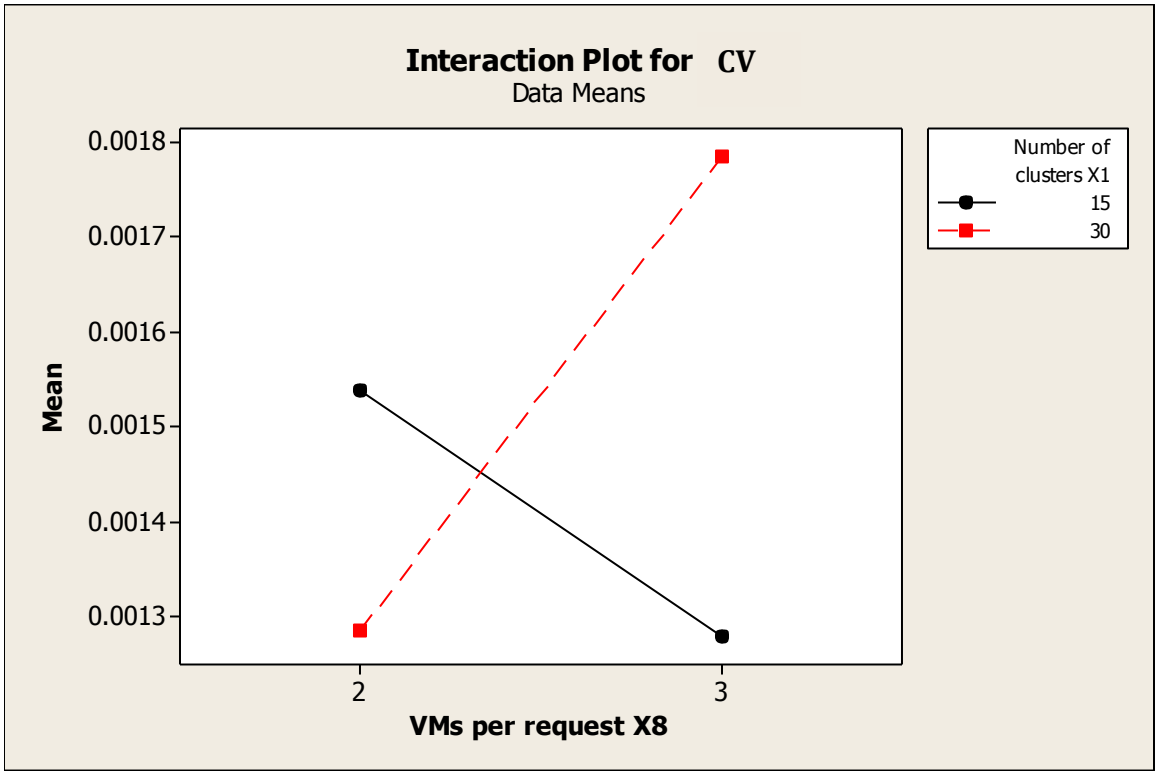


Figure D.22 Interaction plot for *CA* between X1*X8

Appendix E

VMC simulator's Estimates of the Malicious Co-residency Metrics

The following tables (Table E.1 and Table E.2) list the *VMC* simulator's estimates for the malicious co-residency metrics *MCP* and *AFL*. These estimates were obtained under different numbers of hosts and users' arrival rates with an α of 0.10 (see Section 6.2 for the definition of α).

	First Fit		Next Fit		Power Save		Random	
Number of Hosts	**MCP**	**AFL**	**MCP**	**AFL**	**MCP**	**AFL**	**MCP**	**AFL**
1000	0.2306	0.8382	0.3720	0.7919	0.2309	0.8436	0.3765	0.7833
10000	0.2408	0.8221	0.0416	0.9915	0.2372	0.8300	0.0648	0.9682
15000	0.2321	0.8305	0.0087	0.9997	0.3254	0.8087	0.1264	0.9769
30000	0.3182	0.8146	0.0000	1.0	0.3161	0.8151	0.1040	0.9883

Table E.1 The *VMC* simulator's estimates of the malicious co-residency metrics under different numbers of hosts with an α of 0.10

	First Fit		Next Fit		Power Save		Random	
Users' Arrival Rate	**MCP**	**AFL**	**MCP**	**AFL**	**MCP**	**AFL**	**MCP**	**AFL**
2	0.3206	0.8140	0.1864	0.9581	0.3213	0.8116	0.2067	0.9329
3	0.2837	0.8386	0.2537	0.9237	0.2936	0.8351	0.2720	0.8989
4	0.2729	0.8540	0.3224	0.8821	0.2726	0.8559	0.3150	0.8708
5	0.2615	0.8623	0.3685	0.8471	0.2636	0.8648	0.3644	0.8405

Table E.2 The *VMC* simulator's estimates of the malicious co-residency metrics under different users' arrival rates with an α of 0.10

Appendix F
Criteria for Selecting Simulation as a Testbed

F.1 Introduction

As pointed out in Chapter 1, this thesis takes an experimental approach to study co-residency occurrence behaviour in IaaS clouds. Studying co-residency in large, non-transparent and diverse IaaS clouds can become a very challenging task that requires an effective testbed that supports experimentation under different scenarios and settings. Therefore, this appendix makes two contributions. First, a number of cloud platforms and software tools are evaluated on their suitability as experimental testbeds to examine different aspects of co-residency. Second, a comparison is made of different testbeds based on how they meet certain requirements for experimenting on large-scale clouds, such as scalability and cost.

In this appendix, Section F.2 outlines the testbed selection criteria to help identify the most suitable testbed for experiments related to co-residency. In Section F.3, a survey of a number of the available testbeds for experimentation on co-residency is provided, followed by a comparison and evaluation of the elected testbeds according to the selection criteria in Section F.4. This evaluation leads to a recommendation for implementing a new VM Co-residency (*VMC*) simulator in Section F.5. The *VMC* simulator is used in this thesis as a testbed for research on co-residency in the cloud.

It is worth mentioning that a summary of some of the results presented in this appendix appeared in [1].

F.2 Testbed Selection Criteria

It can be argued that there is no single or best approach for experimenting on co-residency in the cloud. This is because too many parameters exist that need to be taken into account when conducting the experiments. Such parameters describe cloud architecture, functional and non-functional requirements.

The thesis's aim (Section 1.3) requires studying co-residency in large IaaS clouds under a variety of settings. Such settings include various cloud user volumes, an assorted number of VMs, different numbers of hosts and clusters and, most importantly, a number of *PA*s.

Performing such an experiment in large and dynamic environments, such as IaaS clouds, needs a testbed that meets a certain set of criteria and requirements. Meeting these requirements helps to conduct the experiment efficiently within the limited time and resources available. For the purpose of conducting this evaluation, it has been assumed that the available time for experimentation on the selected testbed is six calendar months. In addition, the research's available resource is a small lab that consists of four mid-range machines operated by a single researcher. The following selection criteria are set to help choose the most suitable testbed for experiments in this thesis. The first three criteria were inspired by [34]. On the other hand, the remaining criteria were derived from the experience gained from previous own research [2] and were directly relevant to the needs of the type of experiments in this thesis. The final evaluation will examine each testbed against each of the following selection criteria:

F.2.1 Repeatable and Controllable

A repeatable experiment means that re-conducting the same experiment by the same experimenter must produce similar results. Needless to say, being able to conduct and repeat co-residency experiments in unpredictable environment conditions is the most important key to achieving meaningful results. Therefore, conducting and repeating co-residency experiments requires full control of the underlying cloud infrastructure (such as *PA*s, hosts and clusters).

F.2.2 Transparent

It is necessary to have a testbed that offers a safe level of transparency to allow the observation of different aspects of co-residency behaviours, such as detecting co-residency hit and estimating co-residency hit probability.

F.2.3 Flexible

A flexible testbed must easily offer the ability to experiment on several cloud parameters' settings with different levels of details and different *PA*s. This ability is crucial to allow the experiment results to be generalizable.

F.2.4 Accessible

This criterion states that the testbed must be available and legal to use in experimental activities. Also, the time required for downloading, deploying and mastering the testbed with proper technical documentation defines the accessibility requirement.

F.2.5 Scalable

The co-residency experiments in this thesis need to be conducted on various scales of IaaS clouds (such as the various number of hosts, clusters and VMs). Scalability means that the chosen testbed can accommodate the increase in the size of cloud resources while maintaining the minimum expenditure of the research's resources.

F.2.6 Inexpensive and Not Time-Consuming

In general, experimentation on large scale IaaS clouds requires both time and computational resources. It is important to consider the time and budget limitations for running the experiments on the selected testbed. Quick implementations of the experiment on the testbed, with minimum expense, as well as an acceptable execution speed are important factors that influence the testbed selection decisions.

F.2.7 Sufficient Reporting/Monitoring System

Large-scale experiments usually produce a vast amount of output and statistical data that are used to analyse the results. In addition to the need for excellent reporting capabilities, the testbed must also allow the user to monitor effectively and record all necessary actions related to co-residency.

F.3 Available Testbeds

The experimental validation methodologies presented in [34] aim to define the best practices to conduct sound experiments in large-scale systems. The suggested experimental methodologies are categorized based on the type of the testbed they use. They include:

- **Real-platform experiments**: that is executing real applications on real platforms,
- **Benchmarking**: that is executing modelled applications on real platforms,
- **Emulation**: that is executing real applications on modelled platforms and
- **Simulation**: that is executing modelled applications on modelled platforms.

Looking at the above experimental methodologies, the real-platform and the benchmarking experiments usually use real applications/systems as a testbed, whereas the emulation and simulation methodologies use modelled platforms. By focusing on real-platform and simulation methodologies, this thesis uses three different testbeds for comparison based on the aforementioned testbed criteria:

(1) Real public IaaS clouds;
(2) Real private IaaS clouds; and
(3) Simulators.

A straightforward evaluation of the comparison between these three testbeds was conducted to assess each testbed against each criterion. This comparison will help to select the most suitable testbed for the experiments in this thesis.

F.3.1 Public IaaS Clouds

Public IaaS cloud providers offer users the ability to rent computing infrastructure on-demand to cover their needs. Public IaaS clouds such as Microsoft's Windows Azure [57], Amazon's EC2 [4] and Rackspace [77] allow users to run their own VMs (i.e. as servers). In order to utilize their physical infrastructure, virtualization is used to allow physical resources to be shared between users. Because of this, each IaaS cloud exhibits different workloads and can vary in the underlying infrastructure and configurations.

Using public IaaS clouds as testbeds is possible, yet it shows some limitations. [79] pioneered research uses Amazon's EC2 as a testbed. The researchers demonstrate that it is possible to map the internal cloud infrastructure in order to locate and co-reside with targets (see Chapter 2 for more details). They also describe a number of attacking scenarios where a malicious user can gather sensitive information from co-resident VMs that share the same underlying machine using side-channel attacks. Other research, such as the AmazonIA paper [17], have used public IaaS clouds as a testbed. In particular, the researchers in AmazonIA have used Amazon's EC2 to launch various crafted Amazon Image Attacks in which they were able to collect very sensitive information (including credentials, passwords and keys). In addition, Amazon's EC2 also has been used as a testbed in an early stage of this thesis. As explained in Section 2.2, available public IaaS clouds, including Amazon's EC2, are usually accessible and easy to use with their rapid scalability. In addition, public IaaS cloud providers normally supply documentation and how-to-use resources. However, using public

clouds as a testbed comes with its own expenses. The diverse varieties of uncontrollable infrastructure configurations and settings make the use of public IaaS clouds as a testbed in this thesis an unpredictable and time-consuming task. Moreover, the pay-as-you-go nature of the public IaaS cloud and the need for conducting repeatable experiments with different parameters' settings would incur expenses that exceed the available resources. Furthermore, public IaaS cloud providers, such as Amazon EC2 and Windows Azure, usually obscure the details of their cloud infrastructure, networks and even *PA*s, which results in a lack of transparency [79]. With little to no transparency, it becomes difficult to conduct testing experiments on such platforms. This is because the testers cannot obtain the necessary information about the cloud anatomy and the implemented *PA*, making the public IaaS cloud a non-transparent and hard to control testbed. Further, this lack of control might also result in the inability to implement a sufficient reporting system for detecting underlying events related to co-residency. Thus, this lack of control does not support generalizing the experiment's results due to the use of very specific cloud architecture. In some situations, it is also possible that extensive experimental usage might lead to a violation of the cloud's usage policy [4]. From what has been discussed before, this combination of limitations shows that public IaaS clouds are thought not to be always the best testbed for this type of research.

F.3.2 Private IaaS Clouds

Private IaaS clouds, such as the open-source Eucalyptus private cloud [67] and OpenNebula [60] offer similar functionalities as public IaaS clouds. However, there is one major difference: private IaaS clouds are implemented in the user's own physical infrastructure whereas public IaaS clouds run on a third party infrastructure (see Section 2.2). This feature of the private IaaS clouds offers more flexibility to implement and model a vast array of possible cloud architectures. Moreover, an open-source private cloud gives the researchers the necessary transparency to control and monitor every single event in their experiments, which forms a good repeatable and controllable testbed. Also, private IaaS clouds have been used as testbeds in an experimental research context for various objectives. For instance, [6] have conducted an evaluation of software ageing effects on Eucalyptus private cloud infrastructure. Further, other researchers have used Eucalyptus as a proof of concept of autonomic resource provisioning in rocks clusters [43]. However, there is still a need when using private IaaS clouds for large capital investment to purchase and maintain the required

hardware infrastructure to conduct scalable experiments, which can sometimes exceed the available resources for this thesis.

F.3.3 Simulators

One of the widely used testbeds in large-scale experiments is to use simulators, such as grid simulators and cloud computing simulators, instead of using real IaaS clouds as testbeds [34]. Computer simulation refers to the actual running of a program that describes a system model, algorithms or equations.

Continuous simulations mimic physical systems' execution at the exact rate as actual clock time. This is in contrast to discrete-event simulation, which has a collection of state variables that reflect the current system status [12]. These state variables can change only at discrete instants (called events), whose sequential order describes the simulated system behaviour. A list of some of the grid simulators and cloud computing simulators, which are related to the experiments in this thesis, with descriptions and comparisons, is provided next.

F.3.3.1 Grid Simulators

In the area of distributed computing, grid computing is a set of distributed systems that provide on-demand access to dependable, consistent and inexpensive hardware and software infrastructure. Grid computing is usually used to process large amounts of non-interactive workloads [29]. There are many multi-tier data centre simulation platforms that have been designed to support the modelling of different hardware specifications of the common data centres' components. Such components include hosts, network switches and communication links. One example of multi-tier data centre simulators is MDCSim [51]. However, grid simulators require more advanced capabilities in order to simulate the distributed applications' behaviour more accurately. In order to meet the demand of research and development on grid systems, several grid simulators have been introduced. Examples of these simulators include SimGrid [19], MicroGrid [88], GridSim [88] and GangSim [27]. Recently, SimGrid started to support a very basic interface to implement virtualization environments. However, this interface is highly experimental as stated on the project website and that they "…do not expect too much of it right now" [86].

Among these grid simulators, it can be argued that GridSim is the most related to co-residency research as it has been extended to form the base of some of the current cloud simulators [18]. Initially, GridSim was introduced as a simulator for resource modelling,

application scheduling and performance analysis in grid computing environments. It supports the modelling of various application models, and it is capable of automating the task of generating a stream of application workloads. GridSim was built upon SimJava [45], a process-based discrete-event simulation framework implemented in Java. Since SimJava runs a unique thread for each element in the simulation, it has been shown in [68] that SimJava performance degrades when simulating more than 2000 grid entities concurrently. This is due to the high consumption of memory. Since GridSim implements in the exact way in which SimJava simulates the grids, it inherits this scalability limitation. It is important to note that grid simulators have been designed to model comprehensively grid systems to the maximum extent. However, none of these simulators are capable of clearly abstracting the application layer from the virtual and physical machines layer. This type of abstraction is required when trying to model multi-layer architecture such as the IaaS cloud. In addition, the above grid simulators are not initially intended to model virtualized resources (i.e. VMs) [30]. Therefore, it would not be practical to use grid simulators in the co-residency experiments, and therefore cloud simulators are instead considered as better testbeds in this thesis.

F.3.3.2 Cloud Simulators

A cloud simulator is a toolkit that models and simulates different cloud computing elements and environments [58]. Cloud simulators are usually capable of simulating multiple clusters and hosts. In addition, cloud simulators normally model the creation of VMs and the placement of these VMs to hosts. Similarly, cloud simulators usually support the creation of cloud users and the generation of different types of cloud-related events. The use of cloud simulators can provide a higher degree of flexibility to conduct different types of experiments on a close-to-real cloud environment. Several IaaS cloud simulators are reviewed next in order to include them in the evaluation at the end of this appendix.

(i) CloudSim

CloudSim [18] is one of the widely used IaaS cloud modelling and simulation toolkit that was developed at the University of Melbourne, Australia. The main goal of CloudSim is to help IaaS cloud researchers to conduct comprehensive simulation-based experiments. The main features that CloudSim offers includes the modelling and simulation of large-scale IaaS clouds, with configurable data centres, physical hosts, resources and virtualization provisioning, as well as power management. With its multi-layer design framework that

reflects the layered architecture of real IaaS cloud environments, CloudSim was developed using Java and was built on top of the SimJava-based grid simulator GridSim. As described earlier, GridSim has a scalability limitation that CloudSim inherited initially. Therefore, the developers of CloudSim decided to modify the first release of this simulator and implement a new discrete-event management framework. This became the CloudSim core simulation engine (Figure F.1). The new framework uses only three main threaded components, and the remaining entities are implemented as objects. Each component in the CloudSim architecture is implemented as a Java class that can be extended or changed to reflect certain simulation requirements.

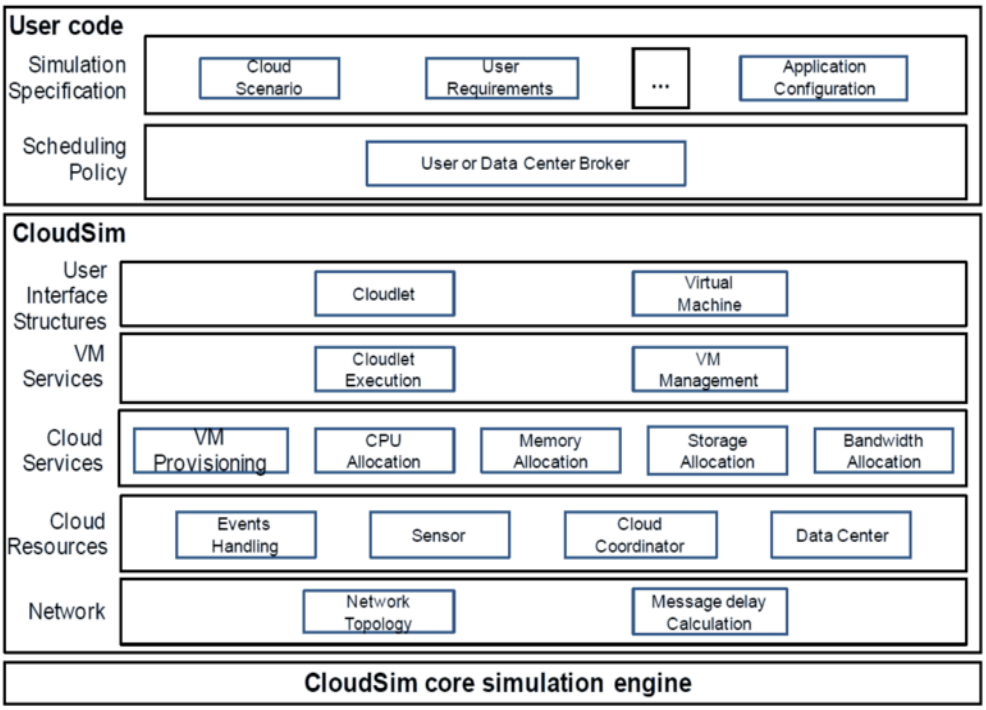


Figure F.1 CloudSim Architecture

Difficulties arise, however, when an attempt is made to simulate certain cloud environments with specific requirements using CloudSim. Each of these different difficulties forms a reason behind the development of many successive simulators that have been built upon CloudSim. At least four cloud computing simulators worldwide have been adopted to extend CloudSim in order to add new functionality or components that CloudSim is missing, such as network latency, bandwidth simulation, SLA management, and more. For example, [39] highlights the need to adopt an easy-to-set-up and user-friendly cloud simulator. They have

surveyed the available cloud simulators in the market and elected CloudSim as a base platform for their intended research. They claim that new enhancements and extensions to CloudSim are essential to maintaining a user-friendly cloud simulator. These extensions have been implemented in the TeachCloud cloud simulator. TeachCloud features a new graphical user interface (GUI) for CloudSim, as well as adding SLA management and business process management modules on the architecture level. In addition, TeachCloud builds several cloud network models such as VL2, BCube, Portland and DCell to model different topologies that can be found in real cloud environments.

Moreover, a group of researchers at the Pontifical Catholic University of Rio Grande do Sul in Brazil have introduced another cloud simulator and visual modeller based on CloudSim, called CloudAnalyst [96]. The primary goals of CloudAnalyst are to visually model, simulate and analyse the effects of geographic distribution of large distributed social network applications under multiple deployment configurations in the cloud. CloudAnalyst gives large applications' developers helpful insights into how to effectively distribute these types of applications. Using CloudSim as the base simulation engine, CloudAnalyst leverages whole features of CloudSim and implements important missing functionalities.

For example, instead of spending unnecessary time on programming the simulation environment requirements using CloudSim, CloudAnalyst provides the user with a GUI to easily control the simulator variables. This action is expected to help the user to focus on the environment simulation experiment. The rest of the added functionality is mainly intended to introduce a basic network, bandwidth and latency modelling management. This allows the user to configure the number of generated applications' workloads, to supply some information of the geographic distribution of the origin of the generating traffic, as well as defining the data centres' locations. By using this detailed information, CloudAnalyst is capable of simulating distributed applications' behaviour in the cloud. Further, CloudAnalyst produces various graphical reports in the form of tables and charts of users' requests response time, requests processing time and other useful analytical data.

In addition, CloudReport [101] is another CloudSim-based cloud computing simulator developed at the Federal University of Ceara, Brazil. Its functionalities are very similar to CloudAnalyst, providing an easy-to-use GUI and a rich reporting module.

Similar to CloudAnalyst, yet with more architecture-level changes, NetworkCloudSim cloud computing simulator [30] has been introduced to overcome the limitations that can be found in CloudSim's network layer. CloudSim's network layer views the data centre's resources as

a collection of VMs, and therefore it is capable of simulating limited communications activities between resources. The developers of NetworkCloudSim argue that CloudSim suffers when simulating a large distributed application (such as message passing parallel applications or multi-tier web applications hosted in different machines). The developers state that a precise evaluation of *PA*s requires a more sophisticated modelling of the data centre's interconnection network. They also claim that they have equipped NetworkCloudSim (Figure F.2) with the most advanced realistic application model compared to CloudSim. Thus, the developers "... have designed a network flow model for Cloud data centres utilizing bandwidth sharing and latencies to enable scalable and fast simulations."

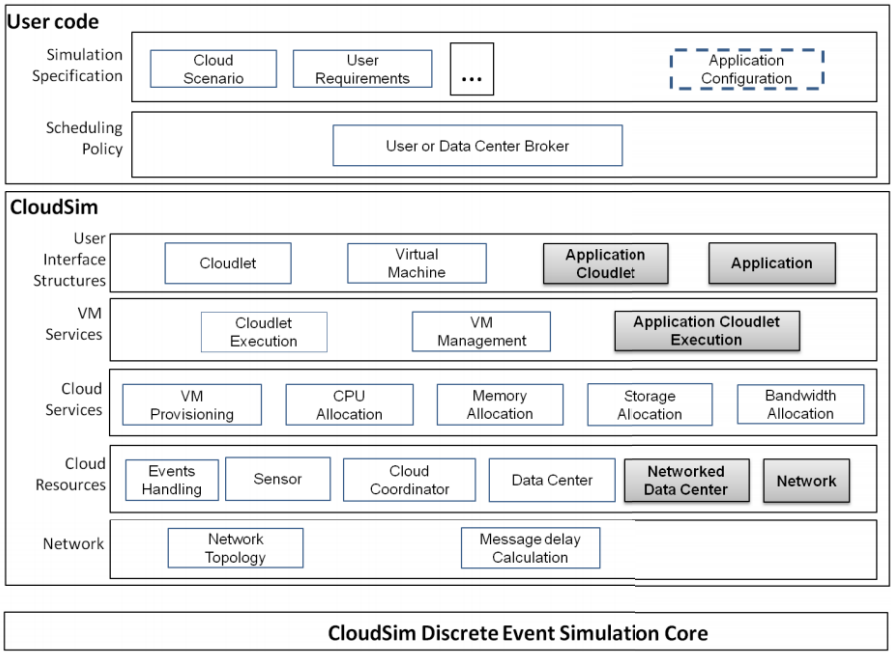


Figure F.2 NetworkCloudSim's new elements introduced to CloudSim Architecture

(ii) GreenCloud

In recent years, there has been an increasing amount of literature on energy-aware cloud data centres. Researchers in this area have started to adopt the use of cloud simulators to experiment with different environment-friendly *PA*s, to utilise the computing resources in an energy-efficient fashion [54]. As an extension of the well-known NS2 network simulator [38], GreenCloud was first introduced in 2010 as a packet-level simulator for energy-aware cloud data centres [44]. Together with the workload generation and distribution which GreenCloud offers, the simulator's primary task is to capture precisely the energy

consumption readings of the data centre components (hosts, switches and links). Moreover, it can simulate and produce the simulation results for two-tier and three-tier architectures. GreenCloud's core strength can be observed in its ability to model the communication interactions of any data centre network with an extensive level of detail since it uses the NS2 to implement a full TCP\IP protocol model. However, this advantage can affect GreenCloud by limiting its scalability due to the heavy memory requirement needed to simulate such detailed models.

(iii) GroudSim

Similar to CloudSim, GroudSim is a Java-based discrete-event cloud simulator developed by [69]. In contrast to CloudSim and the aforementioned cloud simulators, GroudSim is capable of supporting the simulation of applications running on combined cloud and grid platforms. Its developers claim that it offers better scalability and performance compared to related process-based simulators since it uses discrete-event simulation. GroudSim presents some basic analysis and statistics of the simulated cloud. It also supports the modelling of grid and cloud infrastructures including network and computational resources, task scheduling, file transfer, and cost, failure and background models. Nevertheless, GroudSim has not escaped criticism from its developers. They state in [68] that further programming needs to be done in order to implement a different simulation control interface from the one used in the real cloud. This interface is expected to extend the required efforts to execute the simulation experiments.

(iv) Koala

As a medium-scale discrete-event simulation of IaaS clouds, Koala is a project run by the National Institute of Standards and Technology (NIST). The project aims to implement a cloud computing simulator that serves the research on clouds in a more controllable environment [59]. High accuracy models require the definition of many parameters and lead to long run-times resulting in more realistic simulation results, whereas the opposite is true for high abstraction models. Koala has been designed to simulate cloud environments with some abstractions while maintaining a good level of model accuracy. Offering a multi-layered architecture (Figure F.3) based on the commercial discrete-event simulation environment SLX [35], Koala was designed to model the Amazon EC2's architecture through the use of Eucalyptus private cloud APIs.

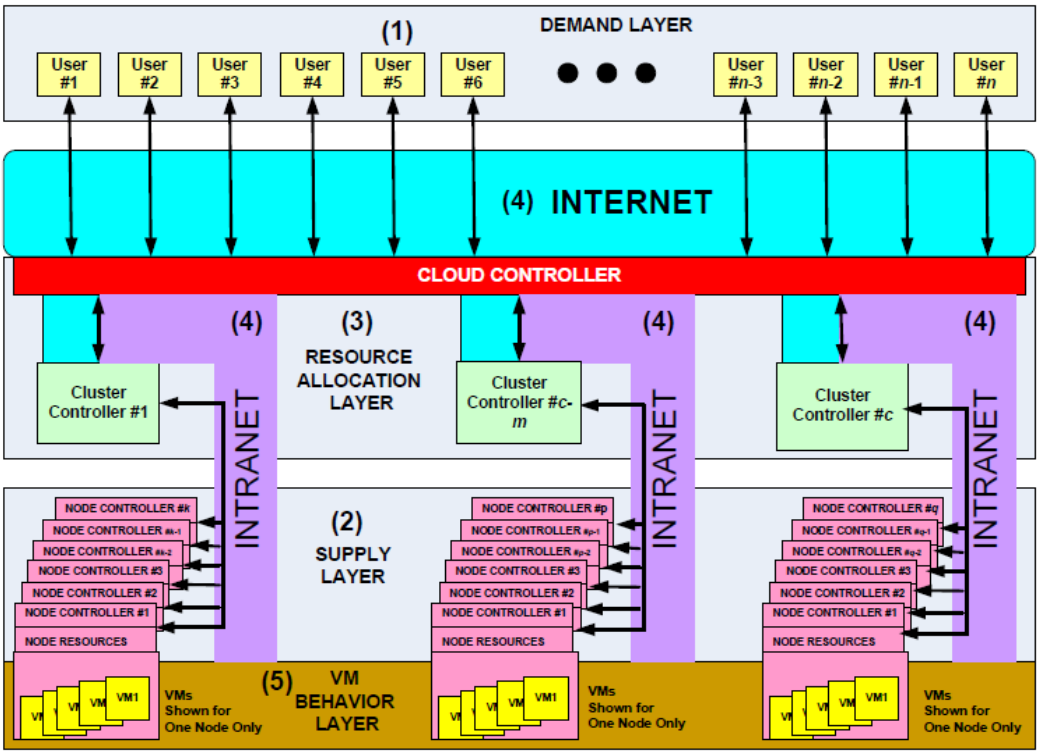


Figure F.3 Koala architecture

Koala is capable of simulating several essential IaaS cloud components, such as cloud controller, cluster controller and host controller, where they all communicate using web services. Initial sensitivity analyses using Koala as a testbed [58] identified the number of cloud users, the number of clusters and number of hosts per cluster as the major parameters that influence the simulator behaviour. Perhaps the most interesting feature of Koala (which has a relation to co-residency's experiments) is that it has several *PA*s already implemented in the cloud controller. These *PA*s are Least-full First, Next Fit, First Fit, Most-full First, Percent Allocated, Random and Tag-and-Pack. Unfortunately, NIST's project would have been more useful for this thesis if Koala's developers had made this simulator available for the researchers to use. This lack of access forms the key issue that might be a strong obstacle that prevents considering Koala as a suitable testbed for co-residency experiments in this thesis.

(v) iCanCloud.

Very much like the Koala simulator, the iCanCloud simulation toolkit was specifically implemented to simulate cloud resources as if they are actually running in the Amazon

Elastic Compute Cloud (EC2). It can also be extended to simulate other IaaS clouds. iCanCloud's primary aim is "… to predict the trade-offs between cost and performance of a given application executed in a specific hardware, and then provide users with useful information about such costs." [66]. Originally built upon the distributed systems simulator, SIMCAN [65], iCanCloud adopts a multi-layer system design that models the common cloud computing stack.

With its user-friendly GUI and the ability to generate graphical reports, iCanCloud simulator easily allows the addition of new cloud components into its repository. Unlike the GroudSim simulator, iCanCloud provides a POSIX-based API for modelling the simulation applications in a much easier way. In addition, Amazon's EC2 is the only environment which is modelled in iCanCloud. However, perhaps the most serious disadvantage of this simulator is that it does not provide a module to take care of creating the cloud resources. Such resources include users, hosts, and VMs, at the start of each simulation run. Instead, it requires the use of the provided GUI to define manually the new cloud resources parameters one by one, which appears to be impractical when modelling a large-scale IaaS cloud environment in this thesis.

F.4 Evaluation and Discussion

The first and foremost decision that need to be made when experimenting with co-residency in the cloud is to select the appropriate testbed that meets the experiment's requirements and constraints. Whether to select a real/physical testbed or a simulator, each option is most suitable in different scenarios and different situations. In this appendix, a number of available testbeds for experimenting on co-residency in the cloud have been surveyed, including real testbeds (i.e. public IaaS and private IaaS clouds) and simulators. A summary of the previous testbeds evaluation is presented in Table F.1.

Criteria	Real Platforms		Simulators	
	Public IaaS	**Private IaaS**	**Grid**	**Cloud**
Repeatable and Controllable	No control on infrastructure and *PA*s	Full Control, runs in local infrastructure		
Transparent	Little to no knowledge about the infrastructure or cloud settings	Full transparency, runs in local infrastructure		
Flexible	Very limited, strict usage policies	Yes, more in open source private clouds	Yes, more in open source simulators	
Accessible	Yes with friendly web-based GUI and instant support	Requires self-maintained infrastructure	Yes when support and documentation are provided	
Scalable	Yes, adding as many resources as needed	Limited hardware infrastructure (e.g. expensive to add more hosts/machines)	Limited in simulators that use threading	
Cost and Time	Pay per use, on demand	Requires investing in physical infrastructure, takes time for deployment and maintenance	Possible to run immediately on a single machine – cost is limited to the license fee (if required) - simulated resources can be added instantly with no cost	
Reporting	Very difficult for co-residency	Requires implementation		

Table F.1 Testbeds evaluation matrix

Simulators are usually capable of modelling several essential IaaS cloud components with some abstractions while maintaining a good level of model accuracy. Simulators can be a sensible option when experimenting on very large-scale and dynamic clouds when there is a need to be able to control and monitor the simulated cloud's behaviour. Section F.3 shows that the aforementioned cloud simulators vary in satisfying the testbed criteria defined earlier. One major criticism is that none of the discussed simulators implements sufficient co-residency monitoring, detection and reporting modules. As stated in Section F.2, these modules are critical when studying co-residency in the cloud. However, implementing these modules into these existing simulators is not an option for closed source simulators. On the other hand, introducing these modules to the open source simulators is possible, but requires a considerable amount of time and effort to achieve. This task becomes more challenging when each of the discussed simulators focuses on modelling cloud elements that are unrelated to co-residency experiments. In addition, some simulators are platform independent (e.g. Java-based simulators) but relatively slow in execution.

Having explored the available public and private IaaS clouds (i.e. the real testbeds) for co-residency experiments, these testbeds usually produce more accurate results than when using simulators, as they are "real" platforms. However, given the context of this thesis, both public and private IaaS clouds have been shown to suffer from a number of shortcomings. For instance, public IaaS clouds are often not transparent testbeds, whereas control and experiment repeatability are hard to achieve. Private IaaS clouds, in particular, can be an expensive option when the experiment needs to be conducted many times on a large and scalable cloud environment. It is worth mentioning that [34] confirms "experiments on real platforms are often not reproducible, whereas, extensibility, applicability and revisability are hard to achieve."

Alternatively, satisfying all testbed criteria in this thesis can be accomplished by designing and implementing a new simulator. This simulator solely implements the system model in this thesis (Section 3.2) and models all the necessary behaviours of co-residency in IaaS clouds. Implementing this new simulator is expected to support the run of this thesis's experiments according to the defined criteria. In fact, implementing and using a purpose-built simulator instead of relying on an existing simulation tool has become a sensible practice for satisfying each individual research's requirements. For instance, [61] analysed 141 research papers that use simulation to study large-scale peer-to-peer systems and reported that 30% of these papers use their own custom simulation tool.

From the previous discussion, simulation experimentation is adopted in this thesis. In using the system model, a purpose-built VM Co-residency (*VMC*) simulator was implemented that allows modelling of co-residency behaviour using various cloud parameters' settings and *PA*s. The next section gives an overview of the *VMC* simulator.

F.5 VM Co-residency (*VMC*) Simulator

Simulating large-scale environments, such as IaaS clouds, can be achieved using several different approaches that aim to provide a controllable, transparent, accessible, scalable and inexpensive test environment. These approaches can be categorised into two main sections: purpose-built (i.e. for a specific use) simulators and general-purpose simulators. Purpose-built simulators usually abstract some components of the modelled environment. On the other hand, this type of simulator carries a very detailed implementation of other components that are more related to the purpose for which the simulator is built [95]. The advantage of using this approach is that the resulting simulator can be rather small in size and, therefore, more scalable as this type of light simulator usually requires less computational resources. However, this imposes some limitations when there is a need to change significantly the system model by changing or adding some of the simulator's missing components. This challenge often involves rewriting a considerable part of the system architecture. On the other hand, general purpose cloud simulators aim to include all possible components of the modelled environments and all intercommunication events [64]. Perhaps the most serious disadvantage of this kind of simulator is that sometimes they do not model enough specifications that are usually required when attempting to conduct precise experimental researches on particular components. In addition, this type of modelling usually results in a large amount of simulation input parameters, which imposes an extra level of complexity when designing the intended simulation experiment [74].

Instead, it is sometimes easier to simulate part of the whole system's components in order to reduce the input parameters. This in turn results in simulating more precise system behaviours and produces more accurate responses [58]. For that reason, many distributed systems simulators, including IaaS cloud simulators, have been purpose-built to simulate specific system architectures or have been implemented only to study certain aspects of the system behaviours. For example, the Koala cloud computing simulator has been specifically designed to model the open-source Eucalyptus IaaS platform structure. Moreover, the

iCanCloud simulator has been specifically modelled to simulate cloud environments as provided by the Amazon EC2 (Section F.3).

Turning now to the *VMC* simulator, the *VMC* has been designed and implemented as a discrete event simulation-based testbed. More attention was paid especially to the provision of comprehensive modelling of *PA*s, co-residency monitoring and detection as well as sufficient reporting modules. This kind of modelling can meet the testbed section criteria (Section F.2) for conducting experiments related to co-residency in particular in an unprecedented fashion. The main reason that motivated the introduction of the *VMC* was to design and implement a simulator that models various IaaS cloud parameters and *PA*s. The *VMC* is expected to assess efficiently the impact of each parameter setting on the co-residency probability. In addition, the *VMC* can be used as an experimentation tool to determine the appropriate parameter settings that reduce the co-residency probability in a given *PA*. The *VMC* has been used successfully as a testbed in this thesis and can be used for future research related to co-residency in IaaS clouds.

With regards to the *VMC* design, the *VMC* has been primarily built as a layered design simulator using object-oriented Java programming language [82], which allowed modularity in the design of the simulator components. This modularity helps to replace easily, reuse or implement more details to the simulator components according to the user needs. This includes modelling distributed clusters with multiple physical hosting machines, different types of cloud users and multiple VMs types. More importantly, the *VMC* implements a number of *PA*s based on the system model used in this thesis (Section 3.2). There are two main reasons behind using layered design. The first is that the Java classes in the multi-layered design enjoy the same module dependency. This module dependency adds an extra level of clarity when looking at the *VMC* design for both simulator designers and users compared to a flat architecture. The second is that the layered design allows a straightforward integration of existing software and tools. Considering the scope of this thesis, the major IaaS cloud elements that are related to this thesis are implemented in *VMC* (Appendix A). On the other hand, some other non-functional elements have been excluded from the current implementation since they are not in the focus of this thesis. Such elements include the cloud services broker, billing management and SLA management. Fortunately, *VMC* modular design, as mentioned earlier, allows the implementation of such components, if needed in the future, easily.

In addition, Appendix A provides more detail on the *VMC* simulator design and implementation as well as the *PA*s that are used in this thesis and how the *VMC* implements them.

F.6 Summary

In this appendix, a number of cloud platforms and software tools have been examined for their suitability as a testbed for experimental research on co-residency. These testbeds have been categorized into real-platforms (i.e. public IaaS clouds and private IaaS clouds) and simulators. These testbeds have been selected based on their popularity, availability of documentation and support, and whether they are applicable for experimental cloud usage. These testbeds were evaluated against seven criteria such as their capabilities and flexibilities in modelling an IaaS cloud, and for input control as well as output analysis. Using simulators can be useful and more effective, especially if real testbeds (public and private IaaS clouds) are expensive or not feasible. However, the evaluation shows that none of the current simulators can be easily utilized for co-residency related research. Therefore, a purpose-built co-residency simulator *VMC* has been implemented and used as a testbed in this thesis. The *VMC* simulator allows the modelling of IaaS cloud environments and also can simulate and monitor the co-residency behaviour in more depth. It is also hoped this co-residency simulator will form a suitable testbed that helps in advancing research on this crucial topic.