PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

UNIVERSITY MOHAMED BOUDIAF - M'SILA

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

DEPARTEMENT OF COMPUTER SCIENCE

N° :……..

DOMAINE : INFORMATICS

FILIERE :SOFTWARE ENGENEERING

OPTION : SOFTWARE ENGENEERING

A dissertation submitted in fulfilment of the requirements for the Degree of Master

By: Bensalah Amin

Subject

On the Performance Evaluation of Cloud Computing Enviroments Metrics using PRISM Model Checker

Presented to the Jury:

.....	University of M'sila	President
Hichem DEBBI ……	University of M'sila	Supervisor
……	University of M'sila	Examiner

Academic year: 2016 /2017

CONTENTS

List of Figures

List of Tables

Generally Introduction

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (for example, networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.[28].

Our work is based on Khazaie's article "A Fine-Grained Performance Model of Cloud Computing Centers" [25] which Quantifies and characterizes such performance measures of Service availability and response time, both of them are important quality measures in cloud's users prospective and realizing both of them requires appropriate modeling.

We propose a PRISM model checker, in this model we are going to implement the analysis modeling by creating sub-models. and the solution is through iteration over individual sub-model solutions, then we try to write the probabilities mentioned in the previous article in CSL language.

The rest of thesis is organized as follows, chapter I introduces cloud computing briefly, and describes the role of virtualization in cloud computing and the procedure of virtualization..Chapter II presents the necessary background information on model checking, including the different probabilistic models such as Discrete-Time Markov Chains and Continuous-Time Markov Chains and the logics PCTL and CSL for property specification. The new elasticity metric and Status, Challenges, Opportunities and the procedure proposed to analysis are defined and described in chapter III. Finally in chapter IV, we investigate the model described in [25], using PISM model checker ,the performance of cloud in different scenarios will be evaluated. At the end, conclusions and future works are listed.

CHAPTER I

CLOUD COMPUTING

1. INTRODUCTION AND BACKGROUND

The idea of providing a centralized computing service dates back to the 1960s, when computing services were provided over a network using mainframe time-sharing technology. In 1966, Canadian engineer Douglass Parkhill published his book The Challenge of the Computer Utility , in which he describes the idea of computing as a public utility with a centralized computing facility to which many remote users connect over networks.

In the 1960s, the mainframe time-sharing mechanism effectively utilized computing resources, and provided acceptable performance to users; however, mainframes were difficult to scale and provision up-front because of increasingly high hardware costs. Accordingly, users didn't have full control over the performance of mainframe applications because it depended on how many users utilized the mainframe at a given moment. As such, with the introduction of personal computers users loved the idea of having full control of their computing resources, even though these resources are not as effectively utilized.

With the change in the semiconductor industry, personal computers became affordable, and business abandoned mainframes. A new challenge was then introduced: how to share the data.

Client-server systems were supposed to address this data-sharing challenge by providing centralized data management and processing servers. As business computing needs grew and the Internet became widely adopted, the initially simple client-server architecture transformed into more complex two-tier, three-tier, and four-tier architectures. As a result, the complexity and management costs of IT infrastructure have skyrocketed even the costs of actual software development in large organizations are typically lower than costs of software and infrastructure maintenance. [1]

Cloud Computing has become one of the most talked about technologies in recent times and has got lots of attention from media as well as analysts because of the opportunities it is offering. The market research and analysis firm IDC suggests that the market for Cloud Computing services was $16billion in 2008 and will rise to $42 billion/year by 2012. It has

been estimated that the cost advantages of Cloud Computing to be three to five times for business applications and more than five times for consumer applications. According to a Gartner press release from June 2008, Cloud Computing will be "no less influential than e-business".

The cloud is a metaphor for the Internet and is an abstraction for the complex infrastructure it conceals. There are some important points in the definition to be discussed regarding Cloud Computing. Cloud Computing differs from traditional computing paradigms as it is scalable, can be encapsulated as an abstract entity which provides different level of services to the clients, driven by economies of scale and the services are dynamically configurable .

Many companies have invested in Cloud Computing technology by building their public clouds, which include Amazon, Google and Microsoft. These companies are often releasing new features and updates of their services. For instance Amazon Web Services (AWS) released a Security2 and Economics3 center on their website to have academic and community advice regarding these issues (Khajeh-Hosseini et al., 2010b, p2). This shows that there are still lots of doubts about the costs and security for enterprises to adopt Cloud Computing. Hence, the issues of economics and security in Cloud Computing for enterprises must be researched [2].

1. Cloud Computing

1.1. Defining Cloud Computing

There are countless definitions and interpretations of cloud computing to be found from multiple sources. In the simplest terms, cloud computing means storing and accessing data and programs over the Internet from a remote location or computer instead of our computer's hard drive. This so called remote location has several properties such as scalability, elasticity etc., which is significantly different from a simple remote machine. The cloud is just a metaphor for the Internet.

When we store data on or run a program from the local computer's hard drive, that is called local storage and computing. For it to be considered cloud computing, we need to access our data or programs over the Internet. The end result is the same; however, with an online connection, cloud computing can be done anywhere, anytime, and by any device.

1.1.1. NIST Definition of Cloud Computing

The formal definition of cloud computing comes from the National Institute of Standards and Technology (NIST): "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. This cloud model is composed of five essential characteristics, three service models, and four deployment models [3].

It means that the computing resource or infrastructure be it server hardware, storage, network, or application software all available from the cloud vendor or provider's site/premises, can be accessible over the Internet from any remote location and by any local computing device. In addition, the usage or accessibility is to cost only to the level of usage to the customers based on their needs and demands, also known as the pay-as-you-goor pay-as-per-usemodel. If the need is more, more quantum computing resources are made available (provisioning with elasticity) by the provider. Minimal management effort implies that at the customer's side, the maintenance of computing systems is very minimal as they will have to look at these tasks only for their local computing devices used for accessing cloud-based resources, not for those computing resources managed at the provider's side. Details of five essential characteristics, three service models, and four deployments.

Many vendors, pundits, and experts refer to NIST, and both the International Standards Organization (ISO) and the Institute of Electrical and Electronics Engineers (IEEE) back the NIST definition.

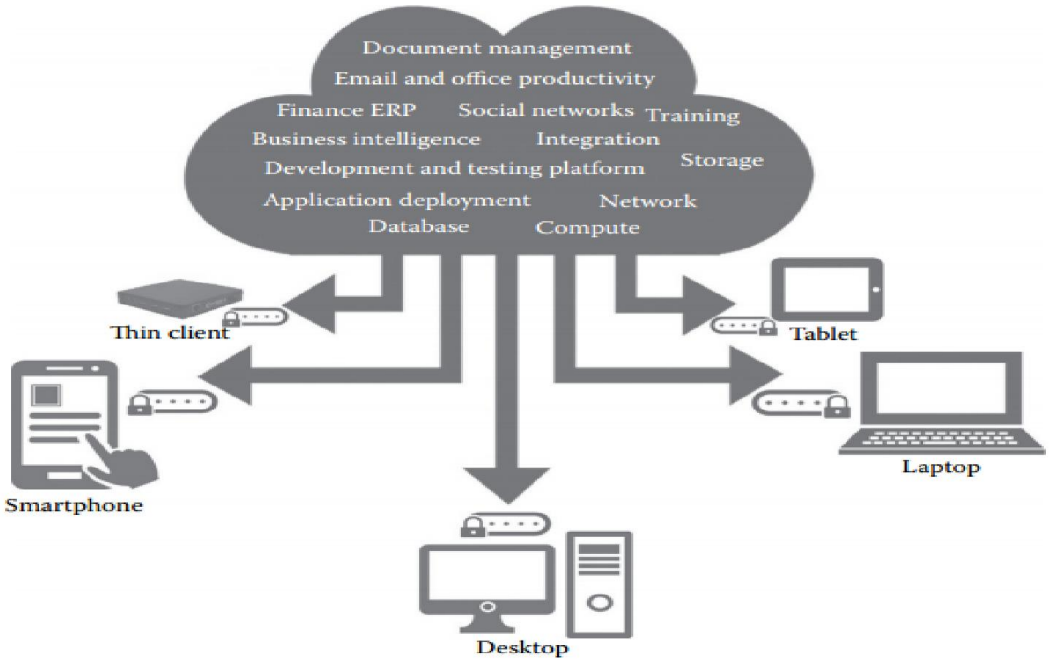


Figure 1.1 paradigm of cloud computing [3].

Now, let us try to define and understand cloud computing from two other perspectives —as a service and a platform in the following sections

1.1.2. CLOUD COMPUTING IS A SERVICE

The simplest thing that any computer does is allow us to store and retrieve information. We can store our family photographs, our favorite songs, or even save movies on it, which is also the most basic service offered by cloud computing. Let us look at the example of a popular application called Flick to illustrate the meaning of this section.

While Flickr started with an emphasis on sharing photos and images, it has emerged as a great place to store those images. In many ways, it is superior to storing the images on your computer:

1. First, Flickr allows us to easily access our images no matter where we are or what type of device we are using. While we might upload the photos of our vacation from our home computer, later, we can easily access them from our laptop at the office.

2. Second, Flickr lets us share the images. There is no need to burn them to a CD or save them on a flash drive. We can just send someone our Flickr address to share these photos or images.

3. Third, Flickr provides data security. By uploading the images to Flickr, we are providing ourselves with data security by creating a backup on the web. And, while it is always best to keep a local copy—either on a computer, a CD, or a flash drive—the truth is that we are far more likely to lose the images that we store locally than Flickr is of losing our images.

1.1.3. CLOUD COMPUTING IS A PLATFORM

The World Wide Web (WWW) can be considered as the operating system for all our Internet-based applications. However, one has to understand that we will always need a local operating system in our computer to access webbased applications.

The basic meaning of the term platformis that it is the support on which applications run or give results to the users. For example, Microsoft Windows is a platform. But, a platform does not have to be an operating system. Java is a platform even though it is not an operating system.

Through cloud computing, the web is becoming a platform. With trends (applications) such as Office 2.0, more and more applications that were originally available on desktop computers are now being converted into web cloud applications. Word processors like Buzzword and office suites like Google Docs are now available in the cloud as their desktop counterparts. All these kinds of trends in providing applications via the cloud are turning cloud computing into a platform or to act as a platform. [3].

1.2. CLOUD COMPUTING ARCHITECTURE

NIST (National Institute of Standards and Technology) is a well accepted institution all over the world for their work in the field of Information Technology. I shall present the working definition provided by NIST of Cloud Computing. NIST defines the Cloud Computing architecture by describing five essential characteristics, three cloud services models and four cloud deployment models .

1.2.1. ESSENTIAL CHARACTERISTICS OF CLOUD COMPUTING

As described above, there are 5 essential characteristics of Cloud Computing which explains there relation and difference from the traditional computing.

• **On-demand-self-service :**Consumer can provision or un-provision the services when needed, without the human interaction with the service provider.

• **Broad Network Access :**It has capabilities over the network and accessed through standard mechanism.

• **Resource Pooling :**The computing resources of the provider are pooled to serve multiple consumers which are using a multi-tenant model, with various physical and virtual resources dynamically assigned, depending on consumer demand.

• **Rapid Elasticity :**Services can be rapidly and elastically provisioned.

• **Measured Service :**Cloud Computing systems automatically control and optimize resource usage by providing a metering capability to the type of services (e.g. storage, processing, bandwidth, or active user accounts)

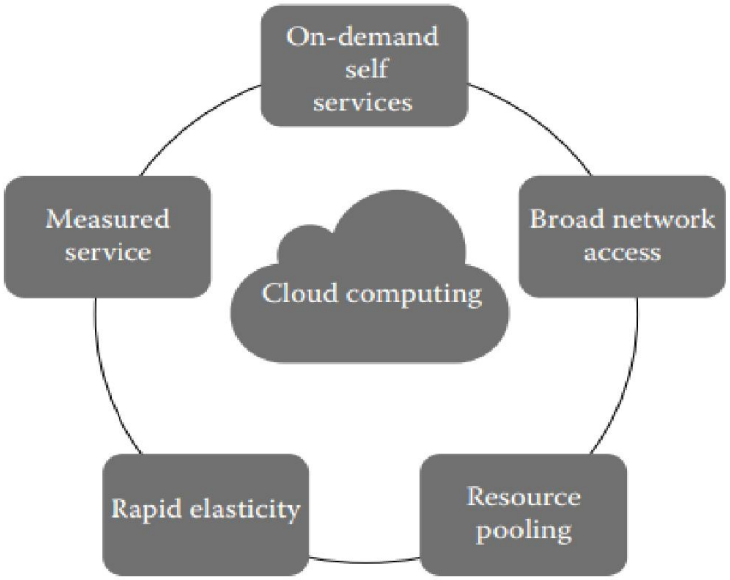


Figure 1.2 The essential characteristics of cloud computing. [3].

1.2.2. CLOUD SERVICE MODELS

There are 3 Cloud Services Models and these 3 fundamental classifications are often referred to as "SPI model" i.e. software, platform or infrastructure as a service .

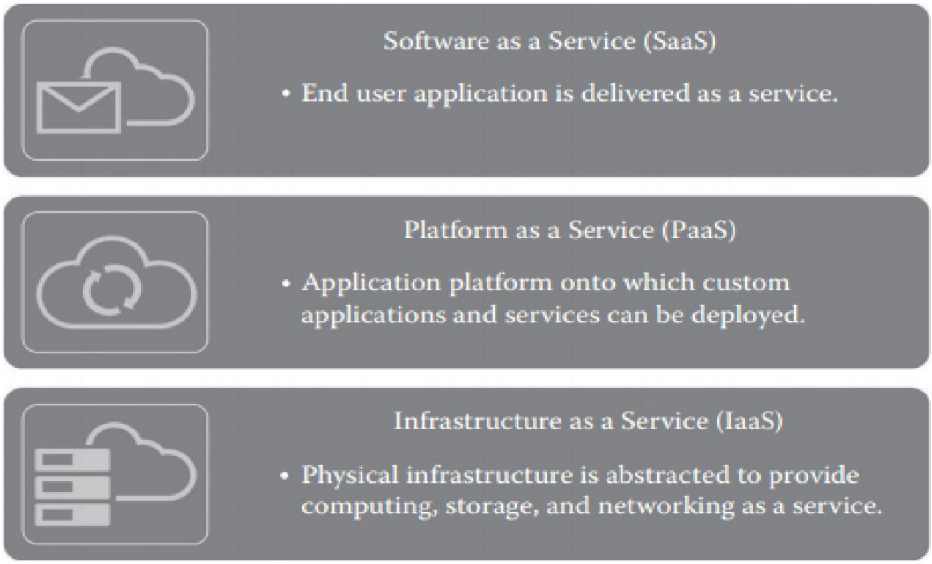


Figure 1.3: SPI—service offering model of the cloud. [3].

• **Cloud Software as Service :**This is a capability in which the consumer can use the provider's applications running on the cloud.

• **Cloud Platform as Service :**In this type of service, the consumer can deploy, the consumer created or acquired applications created by using programming languages or tools provided by provider, on the cloud infrastructure.

• **Cloud Infrastructure as Service :**This is a capability provided to the consumer by which, it can provision processing, storage, networks and other fundamental computing resources where the consumers can deploy and run the software (i.e. operating systems, applications).

1.2.3. CLOUD DEPLOYMENT MODELS

• **Public Cloud :**The cloud infrastructure is available to the general public.

• **Private Cloud :**The type of the cloud, that is available solely for a single organization.

• **Community Cloud :**In this type of cloud deployment model, the infrastructure of the cloud is shared by several organizations and supports a specific community with shared concerns.

• **Hybrid Cloud :**This is a cloud infrastructure that is a composition of two or more clouds i.e. private, community or public .[4]

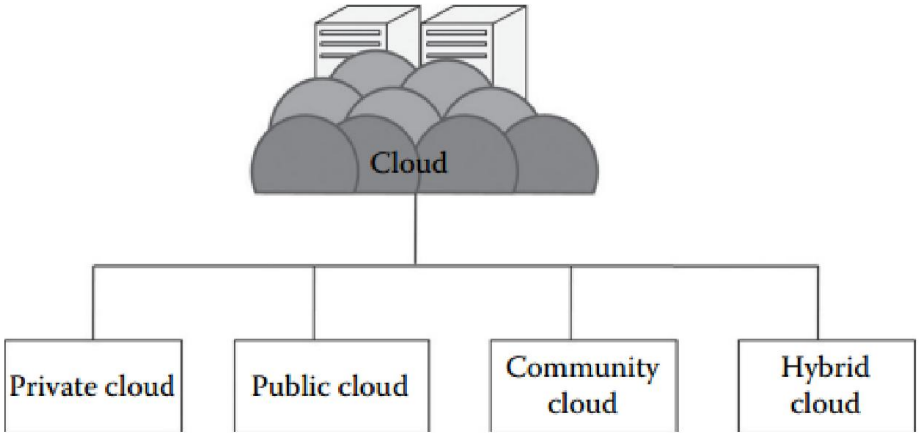


Figure 1.4 Cloud deployment models**.** [3].

2. VIRTUALIZATION

2.1. DEFINING VIRTUALIZATION

Virtualization is a technology that enables the single physical infrastructure to function as a multiple logical infrastructure or resources. Virtualization is not only limited to the hardware, it can take many forms such as memory, processor, network, OS, data, and application. The different forms of virtualization will be discussed in the next section.

Before virtualization, the single physical infrastructure was used to run a single OS and its applications, which results in underutilization of resources. The non shared nature of the hardware forces the organizations to buy a new hardware to meet their additional computing needs.

For example, if any organization wants to experiment or simulate their new idea, they have to use separate dedicated systems for different experiments. So to complete their research work successfully, they tend to buy a new hardware that will increase the CapEx and OpEx. Sometimes, if the organization does not have money to invest more on the additional resources, they may not be able to carry out some valuable experiments because of lack of resources. So, people started thinking about sharing a single infrastructure for multiple purposes in the form of virtualization. The computing scenarios before and after virtualization are shown in Figures 1.5and 1.6, respectively. [3]

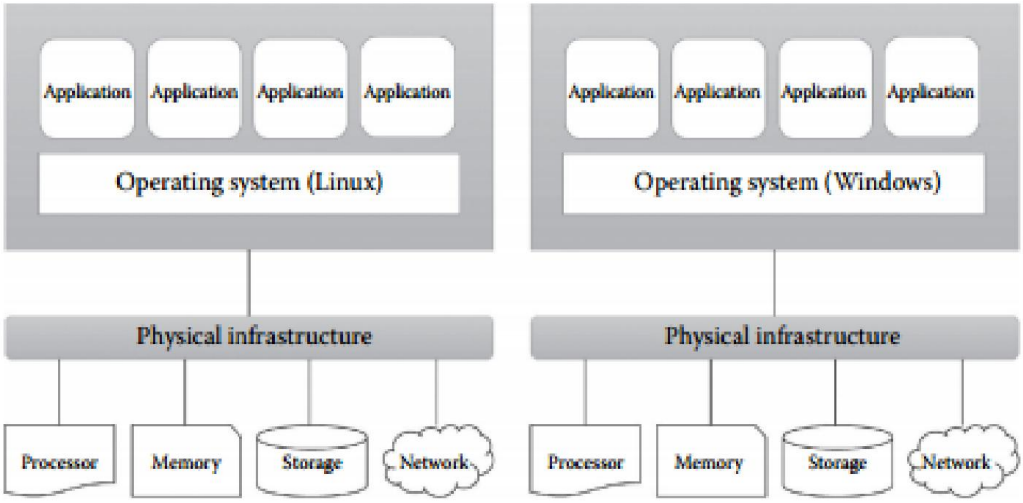


FIGURE 1.5 Before virtualization [3].

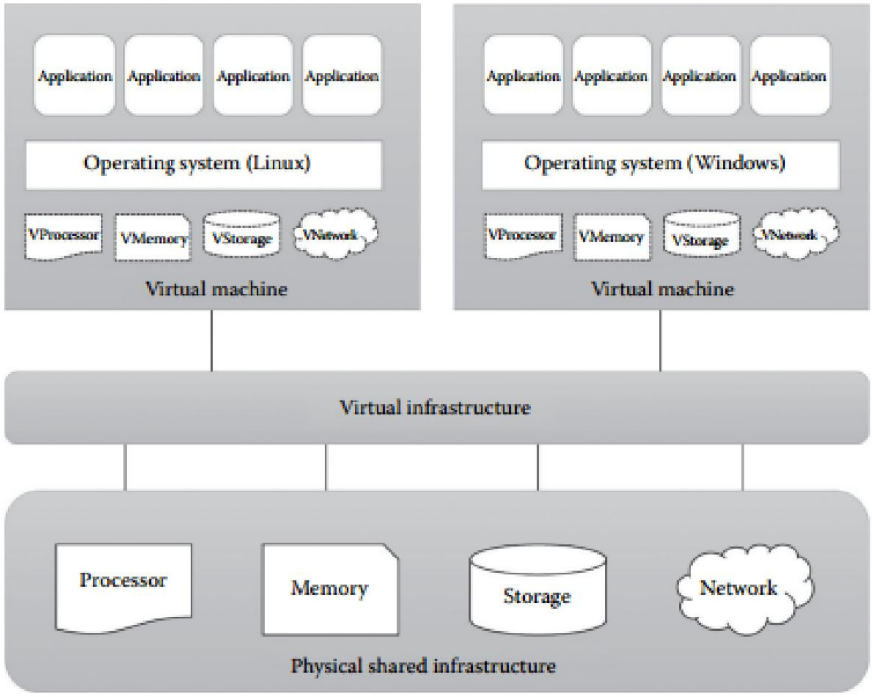


FIGURE 1.6 After virtualization. [3].

After virtualization was introduced, different OSs and applications were able to share a single physical infrastructure. The virtualization reduces the huge amount invested in buying additional resources. The virtualization becomes a key driver in the IT industry, especially in cloud computing. Generally, the terms cloud computing and virtualization are not same. There are significant differences between these two technologies.

Industries adopt virtualization in their organization because of the following benefits:

• Better resource utilization

• Increases ROI

• Dynamic data center

• Supports green IT

• Eases administration

• Improves disaster recovery

While virtualization offers many benefits, it also has some drawbacks:

• Single point of failure

• Demands high-end and powerful infrastructure

• May lead to lower performance .

2.2. VIRTUALIZATION OPPORTUNITIES

Virtualization is the process of abstracting the physical resources to the pool of virtual resources that can be given to any virtual machines (VMs). The different resources like memory, processors, storage, and network can be virtualized using proper virtualization technologies. In this section, we shall discuss some of the resources that can be virtualized.

2.2.1. PROCESSOR VIRTUALIZATION

Processor virtualization allows the VMs to share the virtual processors that are abstracted from the physical processors available at the underlying infrastructure. The virtualization layer abstracts the physical processor to the pool of virtual processors that is shared by the VMs. The virtualization layer will be normally any hypervisors. Processor virtualization from a single hardware is illustrated in Figure1.7 But processor virtualization can also be achieved from distributed servers.

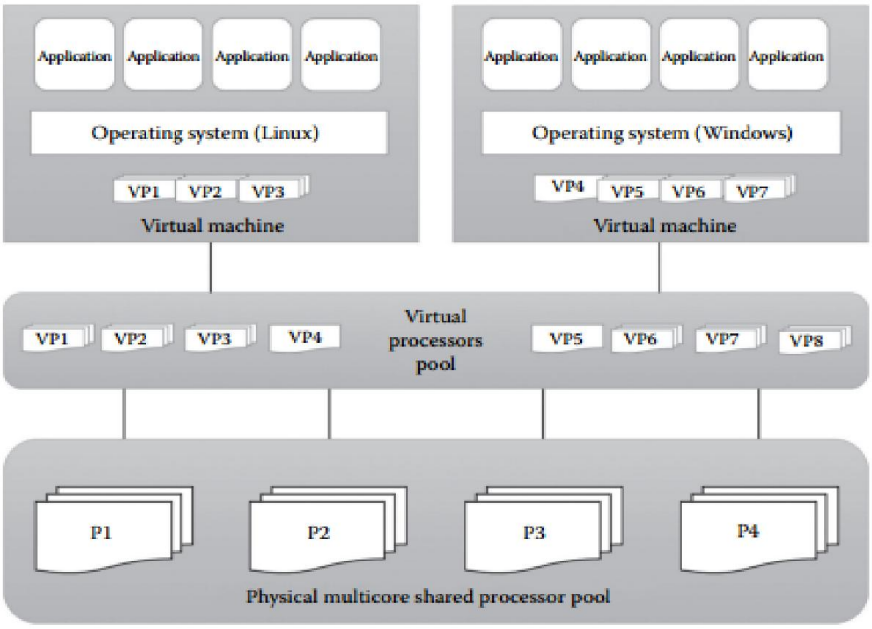


Figure 1.7: Processor virtualization. [3].

2.2.2. MEMORY VIRTUALIZATION

Another important resource virtualization technique is memory virtualization. The process of providing a virtual main memory to the VMs is known as memory virtualization or main memory virtualization. In main memory virtualization, the physical main memory is mapped to the virtual main memory as in the virtual memory concepts in most of the OSs. The main idea of main memory virtualization is to map the virtual page numbers to the physical page numbers. All the modern x86 processors are supporting main memory virtualization. [3]

Main memory virtualization can also be achieved by using the hypervisor software. Normally, in the virtualized data centers, the unused main memory of the different servers will consolidate as a virtual main memory pool and can be given to the VMs. The concept of main memory virtualization is illustrated in Figure 1.8

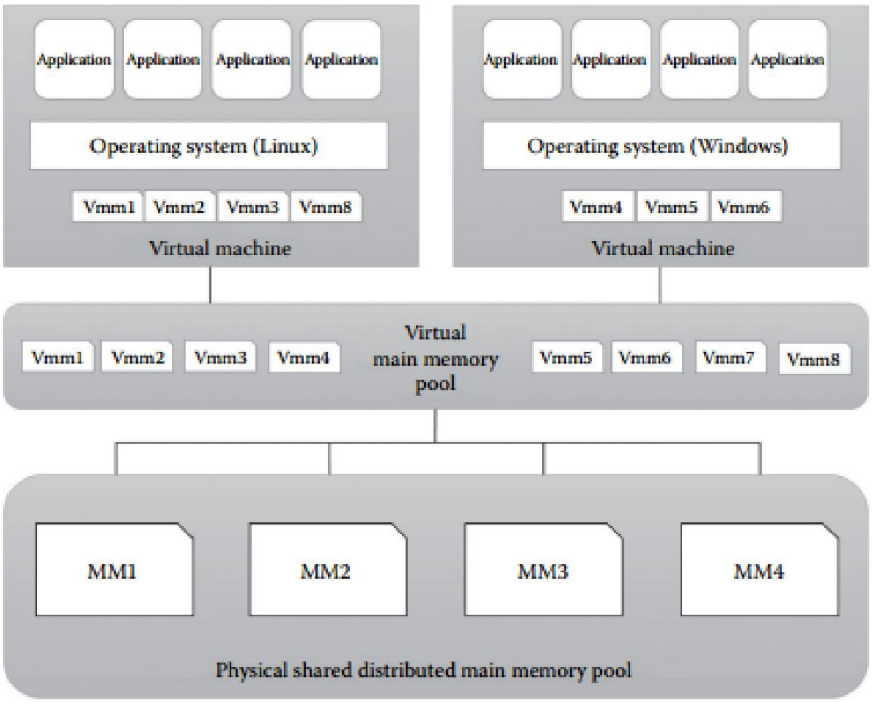


Figure 1.8: Main memory virtualization. [3].

2.2.3. STORAGE VIRTUALIZATION :

Storage virtualization is a form of resource virtualization where multiple physical storage disks are abstracted as a pool of virtual storage disks to the VMs. Normally, the virtualized storage will be called a logical storage. Figure 1.9 illustrates the process of storage virtualization.

Storage virtualization is mainly used for maintaining a backup or replica of the data that are stored on the VMs. It can be further extended to support the high availability of the data. It can also be achieved through the hypervisors. It efficiently utilizes the underlying physical storage. The other advanced storage virtualization techniques are storage area networks (SAN) and network-attached storage (NAS).

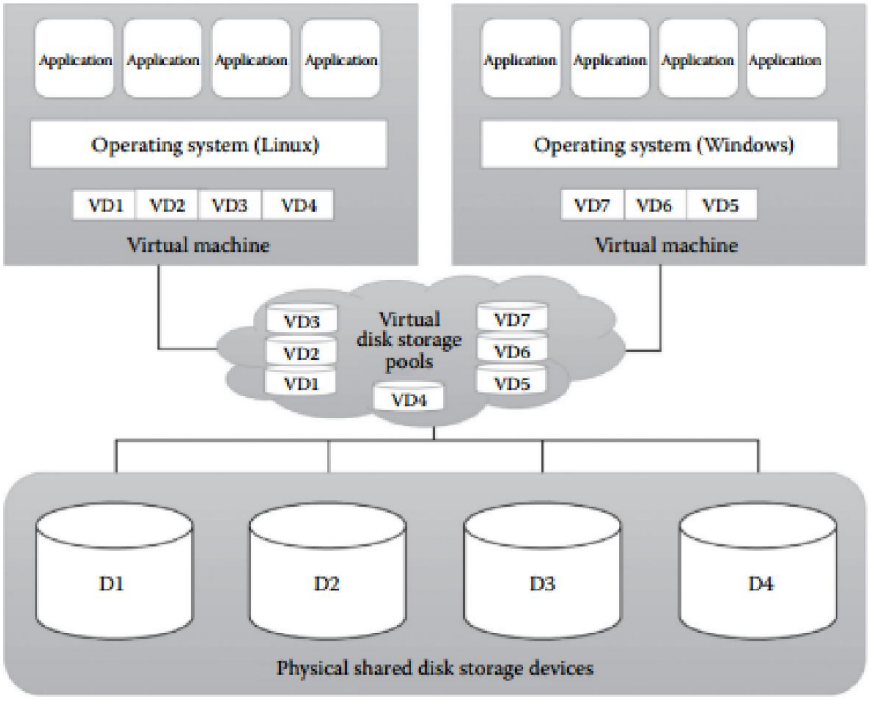


Figure 1.9: Storage virtualization. [3].

2.2.4. NETWORK VIRTUALIZATION

Network virtualization is a type of resource virtualization in which the physical network can be abstracted to create a virtual network. Normally, the physical network components like router, switch, and Network Interface Card (NIC) will be controlled by the virtualization software to provide virtual network components. The virtual network is a single software-based entity that contains the network hardware and software resources. Network virtualization can be achieved from internal network or by combining many external networks. The other advantage of network virtualization is it enables the communication between the VMs that share the physical network. There are different types of network access given to the VMs such as bridged network, network address translation (NAT), and host only. The concept of network virtualization is illustrated in Figure 1.10.

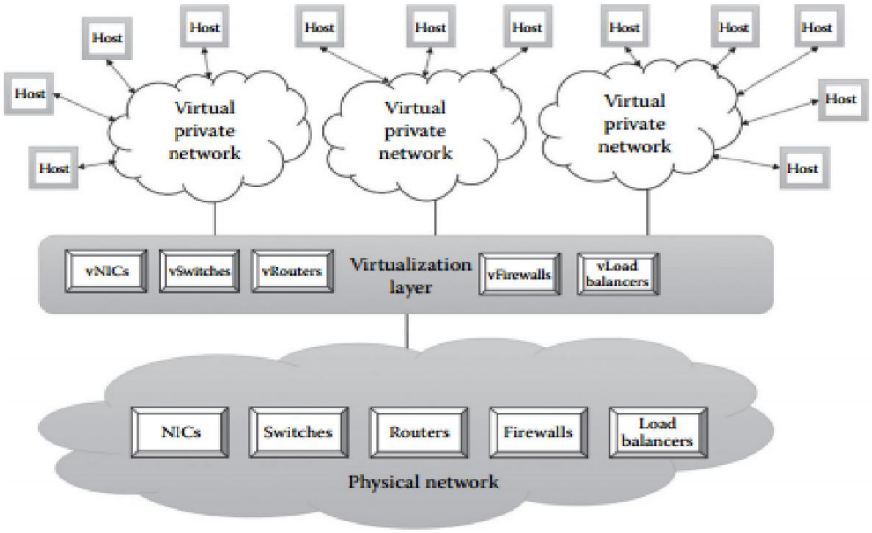


Figure 1.10: Network virtualization**.** [3].

2.2.5. DATA VIRTUALIZATION

Data virtualization is the ability to retrieve the data without knowing its type and the physical location where it is stored. It aggregates the heterogeneous data from the different sources to a single logical/virtual volume of data. This logical data can be accessed from any applications such as web services, E-commerce applications, web portals, Software as a Service (SaaS) applications, and mobile application.

Data virtualization hides the type of the data and the location of the data for the application that access it. It also ensures the single point access to data by aggregating data from different sources. It is mainly used in data integration, business intelligence,and cloud computing. Figure 1.11 represents data virtualization technology.

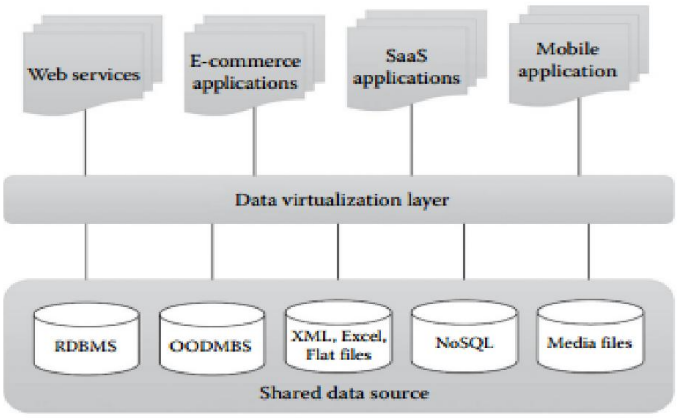


Figure 1.11: data virtualization[3].

2.2.6. APPLICATION VIRTUALIZATION

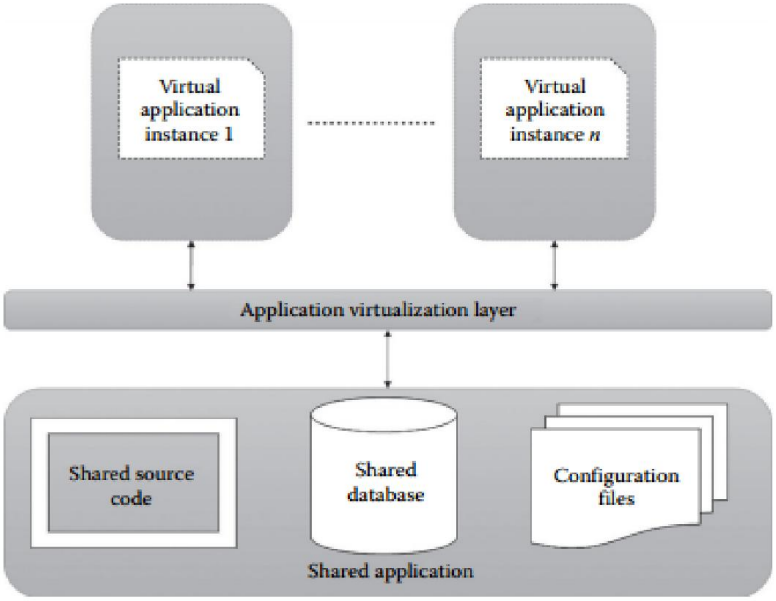


Figure 1.12 : Application Virtualization [3].

Application virtualization is the enabling technology for SaaS of cloud computing. The application virtualization offers the ability to the user to use the application without the need to install any software or tools in the machine. Here, the complexity of installing the client tools or other supported software is reduced. Normally, the applications will be developed and hosted in the central server. The hosted application will be again virtualized, and the users will be given the separated/isolated virtual copy to access. The concept of application virtualization is illustrated in Figure 1.12

2.3. FROM VIRTUALIZATION TO CLOUD COMPUTING

Many users of current IT solutions consider the technologies virtualization and cloud computing as the same. But both technologies are actually different, or in other words, we can say virtualization is not cloud computing. We can prove this claim with the following parameters:

1. Type of service: Generally, virtualization offers more infrastructure services rather than platform and application services. But cloud computing offers all infrastructure (IaaS), platform (PaaS), and software (SaaS) services.

2. Service delivery: The service delivery in cloud computing is on demand and allows the end users to use the cloud services as per the need. But virtualization is not made for on-demand services

3. Service provisioning: In cloud computing, automated and self-service provisioning is possible for the end users, whereas in virtualization, it is not possible and a lot of manual work is required from the providers or system administrator to provide services to the end users.

4. Service orchestration: Cloud computing allows the service orchestration and service composition to meet end user requirements. Some providers are also providing automated service orchestration to the end users. But in virtualization, orchestrating different service to get composite services is not possible.

5. Elasticity: One of the important characteristics that differentiate cloud computing from virtualization is elasticity. In cloud computing, we can add or remove the infrastructure dynamically according to the need, and adding or removing the infrastructure is automatic. But virtualization fails to provide elasticity as stopping and starting a VM is manual and is also difficult.

6. Targeted audience: The targeted audience of these two technologies is also different. Cloud computing targets the service providers for high resource utilization and improved ROI. At the same time, it also facilitates the end users to save money by using on-demand services. In the case of virtualization, the targeted audience is only the service providers or IT owners, not the end users.

2.3.1. IAAS

The cloud computing service delivery model that allows the customers to access the resources as a service from the service provider data center is known as the Infrastructure as a Service (IaaS) model. The virtualization concept is fully utilized in the infrastructure layer of the cloud computing. The IaaS service offers virtual memory, virtual processors, virtual storage, and virtual networks to run the VMs. The general service provisioning mechanism of IaaS services is illustrated in Figure 1.13

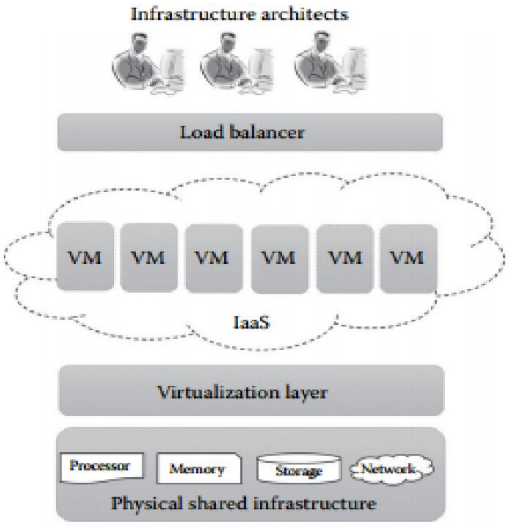


FIGURE 1.13: IaaS. [3].

2.3.2. PAAS

The Platform as a Service (PaaS) allows the end user to develop and deploy the application online by using the virtual development platform provided by the service provider. Generally, the service provider will provide all the development tools as a service to the end users through the Internet. The end users need not install any integrated development environments (IDEs), programming languages, and component libraries in their machine to access the services An overview of a PaaS application is illustrated in Figure 1.14 .

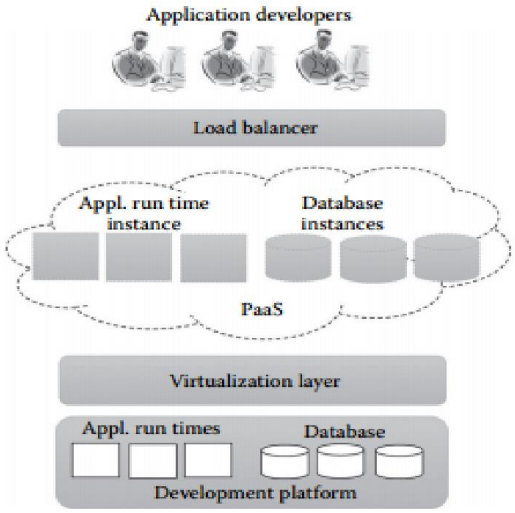


FIGURE 1.14: PaaS [3].

2.3.3. SAAS

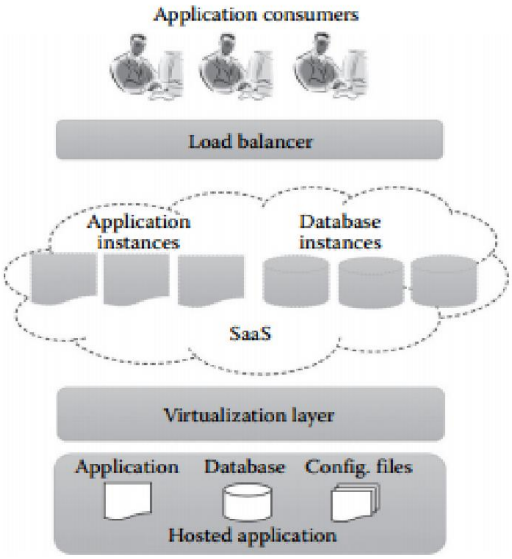


FIGURE 1.15: SaaS [3].

Like infrastructure and platform, software applications can also be virtualized. The software delivery model that allows the customers to access the software that is hosted in the service provider data center through the Internet is known as Software as a Service (SaaS). An overview of a SaaS application is illustrated in Figure 1.15

3. CONCLUSION

Cloud computing is a new paradigm of computing utilities that promises to provide more flexibility, less expensive, and more efficiency in IT services to end users. Firstly this chapter presents an introduction to cloud computing and discuss about on the different types of service models such (IaaS, PaaS, SaaS, NaaS) and Cloud Deployment models.

Chapter II
Model Checking

1. SYSTEM VERIFICATION

System verification techniques are being applied to the design of ICT systems in a more reliable way. Briefly, system verification is used to establish that the design or product under consideration possesses certain properties. The properties to be validated can be quite elementary, e.g., a system should never be able to reach a situation in which no progress can be made (a deadlock scenario), and are mostly obtained from the system's specification. This specification prescribes what the system has to do and what not, and thus constitutes the basis for any verification activity. A defect is found once the system does not fulfill one of the specification's properties. The system is considered to be "correct" whenever it satisfies all properties obtained from its specification. So correctness is always relative to a specification, and is not an absolute property of a system. A schematic view of verification is depicted in Figure 2.1. **[5].**

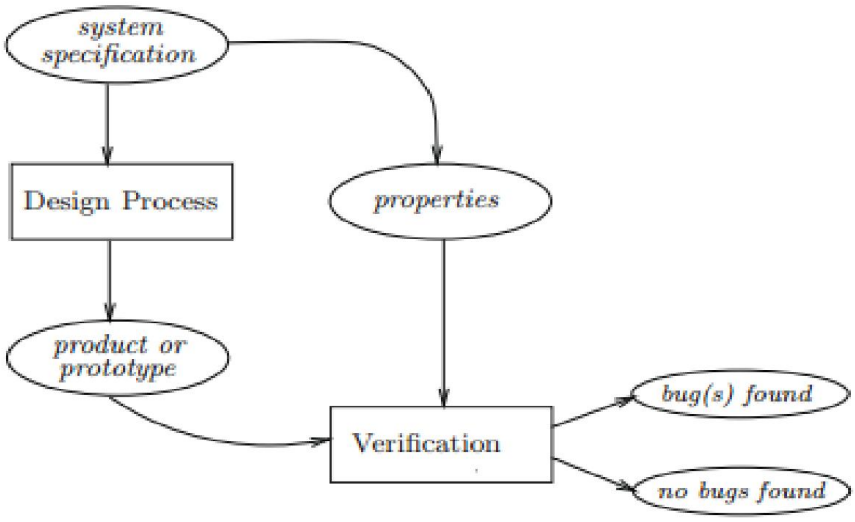


Figure 2.1 Schematic view of an a posteriori system verification **[5].**

This part talk about a verification technique called model checking that starts from a formal system specification. Before introducing this technique and discussing the role of formal specifications, we briefly review alternative software and hardware verification techniques.

1.1. Software Verification:

Peer reviewing and testing are the major software verification techniques used in practice. A peer review amounts to a software inspection carried out by a team of software engineers that preferably has not been involved in the development of the software under review. The un compiled code is not executed, but analyzed completely statically. Empirical studies indicate that peer review provides an effective technique that catches between 31 % and 93 % of the defects with a median around 60%. While mostly applied in a rather ad hoc manner, more dedicated types of peer review procedures, e.g., those that are focused at specific error-detection goals, are even more effective. Despite its almost complete manual nature, peer review is thus a rather useful technique. It is therefore not surprising that some form of peer review is used in almost 80% of all software engineering projects. Due to its static nature, experience has shown that subtle errors such as concurrency and algorithm defects are hard to catch using peer review.

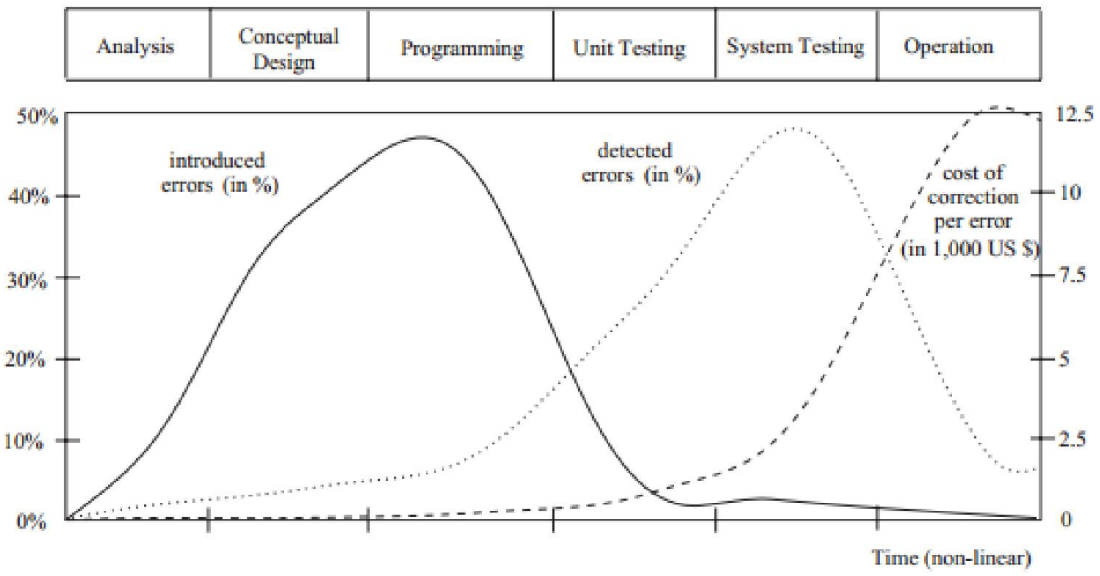


Figure 2.2 Software lifecycle and error introduction, detection, and repair costs **[5].**

1.2. Hardware Verification

Preventing errors in hardware design is vital. Hardware is subject to high fabrication costs; fixing defects after delivery to customers is difficult, and quality expectations are high. Whereas software defects can be repaired by providing users with patches or updates nowadays users even tend to anticipate and accept this hardware bug fixes after delivery to customers are very difficult and mostly require refabrication and redistribution.

Hardware verification techniques Emulation, simulation, and structural analysis are the major techniques used in hardware verification.[5]

2. Model checking

When designing systems (hardware or software) it is important to know whether or not your system will operate as expected/required. For instance you do not want to find out after implementation and delivery of your nuclear power plant control system that it is not as safe as was expected. There are several techniques available that can be used to verify the functional correctness of a system.

Examples of such techniques are: theorem proving, simulation/testing and model checking. namely formal verification by means of model checking . The goal of this technique is to try to predict system behaviour, or more specifically, to formally prove that all possible executions of the system conform to the requirements. Typical problems that are addressed are :

• **Safety** : e. g. does a given mutual exclusion algorithm guarantee mutual exclusion?

• **Liveness** : e. g. will a packet transferred via a routing protocol eventually arrive at the correct destination?

• **Fairness** : e. g. will a repeated attempt to carry out a transaction be eventually granted?

As the name suggests, model checking is performed on a model of a system. The model is usually generated from a high level system description, such as process algebra or Petri net. Typically, these generated models are non-deterministic finite-state automata. These automata (i. e. transition systems) describe the possible system behavior . They can be seen as directed graphs consisting of a finite set of states (nodes) labeled with atomic propositions and state transitions (edges) that show how the system can change from one state to another. The atomic propositions represent the basic properties that hold in each state. Once the system is represented by a model we can check if the model satisfies a formal specification (i. e. if it has certain properties). The properties that are checked against the system model are expressed using logics, such as LTL (Linear Time Logic) or CTL (Computation Tree Logic). These logics can express properties on states or paths in the automata. Once a model and property have been formulated the model checking process will perform a systematic state-space exploration to verify if the property holds.

This form of traditional model checking focuses on delivering a 100% accurate guarantee whether or not a property holds in a certain model. There are many cases where giving an absolute guarantee is not feasible or even impossible. Examples are communication protocols, such as Bluetooth or IEEE 802.11 Wireless LAN, that have to deal with a certain probability of message loss. This is where probabilistic model checking can be utilized. [6]

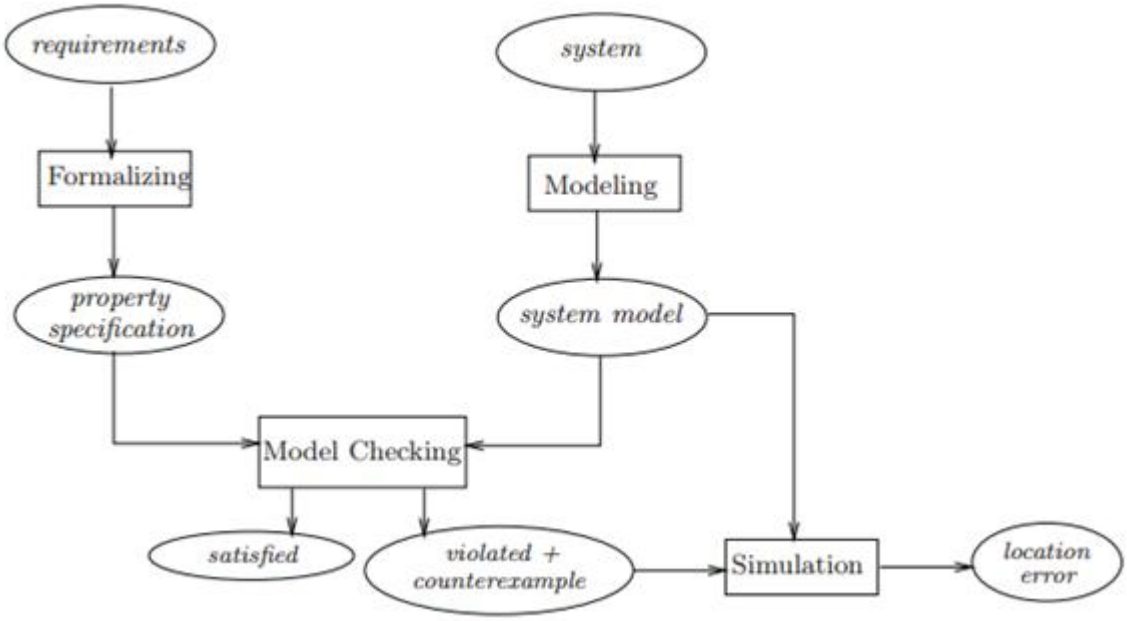


Figure 2.3 Schematic view of the model –checking approach. **[5].**

2.1. The Model-Checking Process

In applying model checking to a design the following different phases can be distinguished:

• Modeling phase:

- model the system under consideration using the model description language of the model checker at hand;
- as a first sanity check and quick assessment of the model perform some simulations;
- formalize the property to be checked using the property specification language.

• Running phase: run the model checker to check the validity of the property in the system model.

• Analysis phase:

- property satisfied?→check next property (if any);

- property violated?→ 1. analyze generated counterexample by simulation;
  2. refine the model, design, or property;
  3. repeat the entire procedure.

- out of memory?→try to reduce the model and try again.

2.2. Advantages of Model Checking

Model Checking has a number of advantages compared to other verification techniques such as automated theorem proving or proof checking. A partial list of some of these advantages is given below:

- No proofs! The user of a Model Checker does not need to construct a correctness proof. In principle, all that is necessary is for the user to enter a description of the circuit or program to be verified and the specification to be checked and press the \ return" key. The checking process is automatic
- Fast. In practice, Model checking is fast compared to other rigorous methods such as the use of a proof checker, which may require months of the user's time working in interactive mode
- Diagnostic counterexamples. If the specification is not satisfied, the Model Checker will produce a counterexample execution trace that shows why the specification does not hold). It is impossible to overestimate the importance of the counterexample feature. The counter examples are invaluable in debugging complex systems. Some people use Model Checking just for this feature.
- Temporal Logics can easily express many of the properties that are needed for reasoning about concurrent systems. This is important because the reason some concurrency property holds is often quite subtle, and it is difficult to verify all possible cases manually

3. Temporal logic

In logic, temporal logic is any system of rules and symbolism for representing, and reasoning about, propositions qualified in terms of time. In a temporal logic we can then express statements like "I am *always* hungry", "I will *eventually* be hungry", or "I will be hungry *until* I eat something". Temporal logic is sometimes also used to refer to tense logic, a particular modal logic-based system of temporal logic introduced by Arthur Prior in the late 1950s, and important results were obtained by Hans Kamp. Subsequently it has been developed further by computer scientists, notably Amir Pnueli, and logicians.

Temporal logic has found an important application in formal verification, where it is used to state requirements of hardware or software systems. For instance, one may wish to say that *whenever* a request is made, access to a resource is *eventually* granted, but it is *never* granted to two requestors simultaneously. Such a statement can conveniently be expressed in a temporal logic.

The modal operators used in Linear Temporal Logic and Computation Tree Logic are defined as follows:

3.1. Linear Temporal Logic: LTL

Temporal logics are a convenient way to formalize and verify properties of reactive systems. LTL is an infinite sequence of states, where each state has a unique succesor. Therefore it establishes a total order on the set of states. Hence one can understand LTL formula's on a line.[7].

LTL is built from the set of atomic propositions (AP), the logical operators ¬ and ∨, and the temporal modal operators **X** and **U**. Formally, the set of LTL formulas over **AP** is inductively defined as follows:

- If $p \in AP$ then p is a valid LTL formula .
- if ψ and ϕ are LTL formulas then ¬ψ, ϕ ∨ ψ, **X** ψ, and ϕ **U** ψ are LTL formulas

X is read as ne**x**t and **U** is read as **u**ntil. Other than these fundamental operators, there are additional logical and temporal operators defined in terms of the fundamental operators to write LTL formulas succinctly. The additional logical operators are ∧, →, ←→, **true**, and **false**. Following are the additional temporal operators.

- **G** for always (**g**lobally)
- **F** for eventually (in the **f**uture)
- **R** for **r**elease
- **W** for **w**eakly until

An LTL formula can be *satisfied* by an infinite sequence of truth evaluations of variables in *AP*. These sequences can be viewed as a word on a path of a Kripke structure (an ω-word over alphabet 2^{AP}). Let $w = a_0,a_1,a_2,...$ be such an ω-word. Let $w(i) = a_i$. Let $w^i = a_i,a_{i+1},...$, which is

a suffix of w. Formally, the satisfaction relation $\models$ between a word and an LTL formula is defined as follows:

- $w \models p$ if $p \in w(0)$
- $w \models \neg\psi$ if $w \not\models \psi$
- $w \models \varphi \vee \psi$ if $w \models \varphi$ or $w \models \psi$
- $w \models \mathbf{X}\,\psi$ if $w^1 \models \psi$ (in the ne**x**t time step ψ must be true)
- $w \models \varphi\ \mathbf{U}\ \psi$ if there exists $i \geq 0$ such that $w^i \models \psi$ and for all $0 \leq k < i$, $w^k \models \varphi$ (φ must remain true **u**ntil ψ becomes true)

We say an ω-word w satisfies an LTL formula ψ when $w \models \psi$. The ω-language $L(\psi)$ defined by ψ is $\{w \mid w \models \psi\}$, which is the set of ω-words that satisfy ψ. A formula ψ is *satisfiable* if there exist an ω-word w such that $w \models \psi$. A formula ψ is *valid* if for each ω-word w over alphabet 2^{AP}, $w \models \psi$.

The additional logical operators are defined as follows:

- $\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$
- $\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$
- $\varphi \leftrightarrow \psi \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
- **true** $\equiv p \vee \neg p$, where $p \in AP$
- **false** $\equiv \neg$**true**

The additional temporal operators **R**, **F**, and **G** are defined as follows:

- $\varphi\ \mathbf{R}\ \psi \equiv \neg(\neg\varphi\ \mathbf{U}\ \neg\psi)$ (ψ remains true until and including once φ becomes true. If φ never become true, ψ must remain true forever.)
- $\mathbf{F}\,\psi \equiv$ **true** $\mathbf{U}\,\psi$ (eventually ψ becomes true)
- $\mathbf{G}\,\psi \equiv$ **false** $\mathbf{R}\,\psi \equiv \neg\mathbf{F}\,\neg\psi$ (ψ always remains true)

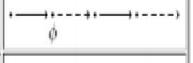
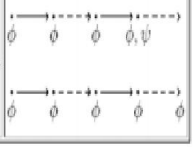
Textual	Symbolic	Explanation	Diagram
Unary operators:			
X ϕ	$\bigcirc\phi$	neXt: ϕ has to hold at the next state.	
G ϕ	$\Box\phi$	Globally: ϕ has to hold on the entire subsequent path.	
F ϕ	$\Diamond\phi$	Finally: ϕ eventually has to hold (somewhere on the subsequent path).	
Binary operators:			
ψ U ϕ	$\psi\mathcal{U}\phi$	Until: ψ has to hold *at least* until ϕ, which holds at the current or a future position.	
ψ R ϕ	$\psi\mathcal{R}\phi$	Release: ϕ has to be true until and including the point where ψ first becomes true; if ψ never becomes true, ϕ must remain true forever.	

Figure 2.4 Ltl operation [7]

3.2. Computational Tree Logic : CTL

Computation Tree Logic (CTL) is a branching-timelogic, meaning that itsmodel of time is a tree-like structure in which the future is not determined;there are different paths in the future, any one of which might be the actual path that is realised.

In CTL, as well as the temporal operators X,F,G and U of LTL we also have quantifiers A and E which express 'all paths' and 'exists a path ,respectively.

Definition

CTL formulas are inductively defined via a Backus Naur form

φ::=> | ⊥ |p|(¬φ)|(φ∧φ)|(φ∨φ)|(φ→φ)|AXφ|EXφ|

(X; U) have the same meaning as in LTL. Where as (A; E) are quantified over path as follows.

- (All): ϕ has to hold on all paths starting from the current state.
- E- (Exists): there exists at least one path starting from the current state where ϕ holds.

Semantics

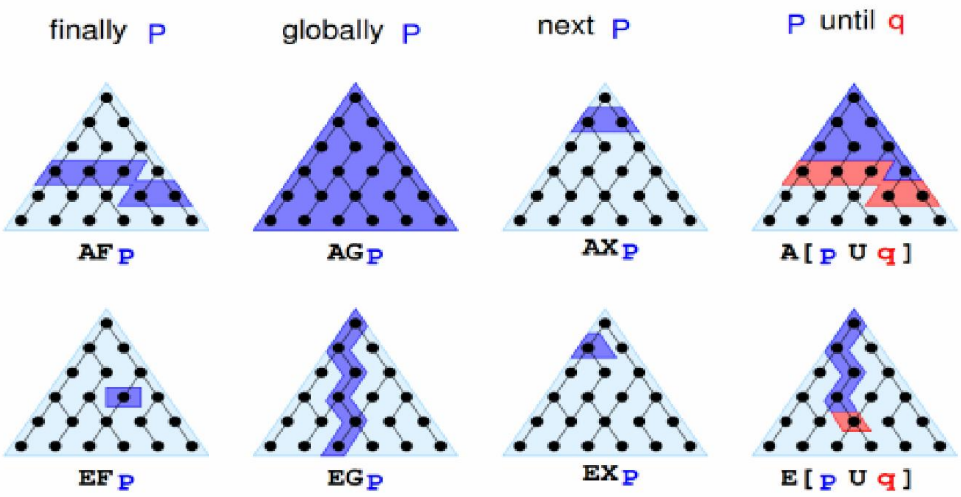


Figure 2.5 Ctl semantic [7]

4. Probabilistic Model Checking

In real life, systems are subject to phenomena of stochastic nature. For instance, it is usually impossible to totally guarantee the correctness of "it is impossible that the process fails" or "a message sent never lost". As a result, we should guarantee instead that " with 0.01 chance the process will fail " or "with chance 0.99 the message will not be lost" . Probabilistic model checking has appeared as an extension of model checking for analysing this kind of systems that exhibit stochastic behaviour. In probabilistic model checking, the model is constructed by assigning probabilities to the transitions between states of the system, and the specifications will be also subjected to deal with probabilities thresholds.

In all information and communication systems, Markov chains have been proved as the most probabilistic operational model, in probabilistic model checking are also the most used. Roughly speaking, Markov chains are transition systems with probability distributions over the states. In probabilistic model checking , the probabilistic systems are usually described using Discrete-Time Markov Chains (DTMC) or ContinuousTime Markov Chains (CTMC) , and Markov Decision Processes (MDP) for non-diterminismtic systems, and verified against properties specified in Probabilistic Computation Tree Logic (PCTL) or Continuous Stochastic Logic (CSL) . While we use PCTL for specifying properties of DTMCS, we use CSL for specifying properties of CTMCs. probabilistic model checking process like conventional model checking consists of the same steps, modelling, specification and verification .[8]

5. Markov chain

A Markov chain is a stochastic process with the Markov property. The term "Markov chain" refers to the sequence of random variables such a process moves through, with the Markov property defining serial dependence only between adjacent periods (as in a "chain"). It can thus be used for describing systems that follow a chain of linked events, where what happens next depends only on the current state of the system.

The system's state space and time parameter index needs to be specified. The following table gives an overview of the different instances of Markov processes for different levels of state space generality and for discrete time vs. continuous time:[wiki]

	Countable state space	**Continuous or general state space**
Discrete-time	(discrete-time) Markov chain on a countable or finite state space	Harris chain (Markov chain on a general state space)
Continuous-time	Continuous-time Markov process or Markov jump process	Any continuous stochastic process with the Markov property, e.g. the Wiener process

Table 1. : different instances of Markov processes

5.1. Probabilistic Models:

Definition. (Discrete Time Markov Chain (DTMC)) A Discrete-Time Markov Chain (DTMC) is a tuple $D = (S, s_{init}, P , L)$, such that S is a finite set of states, $s_{init} \in S$ the initial state, $P : S \times S \rightarrow [0, 1]$ represents the transition probability matrix, $L : S \rightarrow 2^{AP}$ is a labeling function that assigns to each state $s \in S$ the set L(s) of atomic propositions.

Definition (Continuous Time Markov Chain (CTMC)) A Continuous Time Markov

Chain (CTMC) is a tuple $C = (S, s_{init}, \Re, L)$, such that S is a finite set of states, $s_{init} \in S$ the initial state, $\Re : S \times S \rightarrow R>0$ represents the transition rate matrix, $L : S \rightarrow 2^{AP}$ is a labeling function that assigns to each state $s \in S$ the set L(s) of atomic propositions

Comparing to DTMC, the main difference is that with DTMC we have the transition probability matrix that corresponds to discrete-time steps, whereas with CTMC, the transition can occur in real-time, and thus are presented by the transition rate matrix, where every time rate of transition from s to s′is given by $\Re(s, s')$. This parameter represents a negative exponential distribution that contributes to computing the transition probability withint time t units.

Definition (Markov Decision Process (MDP)) A Markov Decision Process (MDP)

is a tuple $M = (S, s_{init}, A, P , L)$, where S is a finite set of states, s init ∈ S is the initial state, A is a set of actions , $P : S \times A \times S \rightarrow [0, 1]$ is a probability transition function such that for every state s ∈ S and an action α ∈ A : $\sum s' \in S\ P(s, \alpha , s') \in \{0, 1\}$, and $L : S \rightarrow 2^{AP}$ is a labeling function that assigns to each state s ∈ S a set of atomic propositions

5.2. Logics for checking probabilistic models

Once the system is represented by a model we want to check if the model satisfies a formal specification (i. e. if it has certain properties). These properties can be expressed in a formal manner using logics. These logics enable us to reason about qualitative or quantitative properties of probabilistic systems. This section discusses two temporal logics, namely Probabilistic Computation Tree Logic (PCTL) and Continuous Stochastic Logic (CSL) ,which are used for verification of DTMCs and CTMCs respectively.

5.2.1. Probabilistic Computation Tree Logic (PCTL)

The Probabilistic Computation Tree Logic (PCTL) .has appeared as an extension of CTL for the specification of systems that exhibit stochastic behaviour. We use the PCTL for defining quantitative properties of DTMCs. PCTL state formulas are formed according to the following grammar:

$\varphi ::= true|a|\neg\varphi\ |\varphi 1 \wedge \varphi 2|P\sim p(\phi\ )$

Where a ∈ AP is an atomic proposition, ϕ is a path formula, P is a probability threshold operator, ∼∈ {<, ≤,>, ≥} is a comparison operator, and p is a probability threshold. The path formulas ϕ are formed according to the following grammar:

$\phi ::= \varphi 1 U \varphi 2|\varphi 1 W \varphi 2|\varphi 1 U{\leq}n\varphi 2|\varphi 1 W{\leq}n\varphi$

Where φ_1 and φ_2 are state formulas and $n \in N$. As in CTL, the temporal operators (U for strong until, W for weak (unless) until and their bounded variants) are required to be immediately preceded by the operator P. The PCTL formula is a state formula, where path formulas only occur inside the operator P. The operator P can be seen as a quantification operator for both the operators ∀ (universal quantification) and ∃ (existential quantification), since the properties are representing quantitative requirements.

The semantics of a PCTL formula over a state s (or a path σ) in a DTMC model D = (S, s init , P , L) can be defined by a satisfaction relation denoted by | =. The satisfaction of P∼p (ϕ) on DTMC depends on the probability mass of set of paths satisfying ϕ . This set is considered as a countable union of cylinder sets, so that, its measurability is ensured.

The semantics of PCTL state formulas for DTMC is defined as follows:

$s \models true \Leftrightarrow true$

$s \models a \Leftrightarrow a \in L(s)$

$s \models \neg\varphi \Leftrightarrow s \not\models \varphi$

$s \models \varphi 1 \wedge \varphi 2 \Leftrightarrow s \models \varphi 1 \wedge s \models \varphi 2$

$s \models P{\sim}p(\phi) \Leftrightarrow P(s \models \phi) \sim p$

Given a path σ = s0 s1... in D and an integer $j \geq 0$, where σ [j] = s j , The semantics of PCTL path formulas for DTMC is defined as for CTL as follows:

$\sigma \models \varphi 1 U \varphi 2 \Leftrightarrow \exists\, j \geq 0.\sigma[j] \models \varphi 2 \wedge (\forall 0 \leq k < j.\sigma[k] \models \varphi 1)$

$\sigma \models \varphi 1 W \varphi 2 \Leftrightarrow \sigma \models \varphi 1 U \varphi 2 \vee (\forall k \geq 0.\sigma[k] \models \varphi 1)$

$\sigma \models \varphi 1 U{\leq}n \varphi 2 \Leftrightarrow \exists 0 \leq j \leq n.\sigma[j] \models \varphi 2 \wedge (\forall 0 \leq k < j.\sigma[k] \models \varphi 1)$

$\sigma \models \varphi 1 W{\leq}n \varphi 2 \Leftrightarrow \sigma \models \varphi 1 U{\leq}n \varphi 2 \vee (\forall 0 \leq k \leq n.\sigma[k] \models \varphi 1)$

The satisfaction of P∼p (ϕ) on DTMC depends on the probability mass of set of paths satisfying ϕ . This set is considered as a countable union of cylinder sets, so that, its measurability is ensured. A formula P∼p (ϕ) is satisfied on an MDP M if only if for every d ∈ D:

P(ϕ)~ p, where D represents the set of all schedulers and P(ϕ) represents the probability of the set of all finite paths satisfying ϕ under the scheduler d . The semantics of PCTL state and of path formulas for MDPs are defined as the same as for DTMCs, except that for model checking of MDPs we have to consider either maximizing or minimizing schedulers. let P max(ϕ) be the maximal probability of ϕ where P max(ϕ) =max{Pd(ϕ)|d ∈ D}, and dually the minimal probability P min(ϕ) be the minimal probabilityof ϕ where P min(ϕ) = min{Pd(ϕ)|d ∈ D}. For instance for properties of upper threshold , itis evident that (M⊨ = P ≤p(ϕ)) ⇔ Pmax(ϕ)>p.

5.2.2. Continuous Stochastic Logic (CSL)

Continuous Stochastic Logic (CSL) was originally developed by Aziz et al. [9] and later extended by Baier et al. [10]. It is based on the temporal logics CTL [11] and PCTL. It provides means to express properties on CTMCs that refer to steady-state and transient behavior . CSL resembles PCTL, in fact it extends PCTL, however the difference lies in the time domain. PCTL is restricted to step intervals of natural numbers N, whereas CSL allows real numbers greater than or equal to zero $R_{\geq 0}$.

Definition (CSL syntax, adapted from [12])**.** Let p ∈ [0, 1] be a real number, I ⊆ $R_{\geq 0}$ a non-empty interval and ./ ∈ {<, >, ≤, ≥} a comparison operator. The syntax of CSL formulas over a set of atomic propositions AP is defined as follows:

- true is a state-formula,
- Each atomic proposition a ∈ AP is a state formula,
- If φ and ψ are state formula, then so are ¬φ and φ ∧ ψ,
- If φ is a state formula, then so is $\mathbf{S}_{\times p}(\varphi)$,
- If Ψ is a path formula, then $\mathbf{P}_{\times p}(\Psi)$ is a state formula,
- If φ and ψ are state formulas, then $\mathbf{X}^{I}$ φ and φ **U** ψ and φ $\mathbf{U}^{I}$ ψ are path formulas.

The state formulas do not differ from those used in PCTL, except for the steady-state $\mathbf{S}_{./p}(\varphi)$ which corresponds to the long-run operator $\mathbf{L}_{./p}(\varphi)$. It asserts that the probability of being in a φ state on the long run, meets the bound ./ p. The path formula $\mathbf{X}^{I}$ φ asserts that a transition is made to a φ state at some time point t ∈ I. Formula φ $\mathbf{U}^{I}$ ψ states that ψ is satisfied

at some time instant t, within the interval I and at all preceding time instants [0, t) φ holds. The unbounded until operator **U** is another notion for asserting φ $\mathbf{U}^{[0,\infty]}$ ψ. We use a satisfaction relation $\models_{\mathbf{M}}$ to define the truth of CSL formulas, for state s and path π in a CTMC $\mathcal{M}$ = (S, s, R, L).

Definition (CSL semantics)**.** Let p ∈ [0, 1] be a real number, and t ∈ R ,a comparison operator. Also let π[i] = s_i be the **i** th state along the path π. Let δ(π, i) = t_i be the time spent in state s_i , and let π@t denote the state occupied in path π at time t. (Similar definitions can be found in [10, 12]) The satisfaction relation $\models_{\mathcal{M}}$, where s is a state, π a path and M a CTMC, is defined by:

$$
\begin{array}{ll}
s \models_{\mathcal{M}} true & \text{for all states,} \\
s \models_{\mathcal{M}} a & \text{iff } a \text{ is an atomic proposition valid in s, } a \in Label(s), \\
s \models_{\mathcal{M}} \neg\phi & \text{iff } s \not\models_{\mathcal{M}} \phi, \\
s \models_{\mathcal{M}} \phi \wedge \psi & \text{iff } s \models_{\mathcal{M}} \phi \wedge s \models_{\mathcal{M}} \psi, \\
s \models_{\mathcal{M}} \mathcal{S}_{\bowtie p}(\phi) & \text{iff } \lim_{t\to\infty} Pr_s\{\pi \in Path^{\mathcal{M}}(s) \mid \pi@t \models_{\mathcal{M}} \phi\}, \\
s \models_{\mathcal{M}} \mathcal{P}_{\bowtie p}(\Psi) & \text{iff } Pr_s\{\pi \in Path^{\mathcal{M}}(s) \mid \pi \models_{\mathcal{M}} \Psi\} \bowtie p, \\
\pi \models_{\mathcal{M}} \mathcal{X}^I \phi & \text{iff } \pi[1] \text{ is defined and } \pi[1] \models_{\mathcal{M}} \phi \ \wedge\ \delta(\pi, 0) \in I, \\
\pi \models_{\mathcal{M}} \phi\, \mathcal{U}^I\, \psi & \text{iff } \exists t \in I.\ (\pi@t \models_{\mathcal{M}} \psi \ \wedge\ (\forall t' \in [0, t).\pi@t' \models_{\mathcal{M}} \phi)).
\end{array}
$$

6. PRISM model checker

PRISM is a probabilistic model checking tool developed at Oxford University Computing Laboratory. A notable difference from the other model checking framework frameworks is that it can construct system models exhibiting probabilistic behavior and check quantitative properties specified in Continuous Stochastic Logic (CSL) language, which is the an extension of CTL with probabilistic operators.

PRISM can build and analyse several types of probabilistic models:

- discrete-time Markov chains (DTMCs)
- continuous-time Markov chains (CTMCs)
- Markov decision processes (MDPs)
- probabilistic automata (PAs)
- probabilistic timed automata (PTAs)

Here we show an example of probabilistic model verification using the PRISM framework. We consider a PRISM state transition model expressing the behavior of the queuing system depicted in Figure 2.6. In this model, we apply the following assumptions. First, the request arrival rate is λ and the service rate is μ . Next, the distributions of the service rate and the interval between request arrivals are exponential. Then, the maximum queue length is 3. Figure 2.7 shows the Continuous-time Markov chain (CTMC) state transition model representing the behavior of the queuing system. In this model, there are four states corresponding to the queue length. The arrival of request at rate λ is represented by the state transitions from the state where the queue length is q = n to the state where q = n + 1 . These transitions are evoked at the rate λ . Likewise, departures from the queue are represented by the opposite state transitions at the rate μ . We can describe this CTMC state transition model in PRISM language as shown in fig 2.8 . The system is modeled by a module in which we define two types of transitions with labels [Arrival] and [Departure] , respectively. These transition definitions mean that the parameter q representing the queue length at each state increases at the rate lambda and decreases at the rate mu. We can synchronize the state transitions in several modules by defining the state transitions with the same label in different modules.

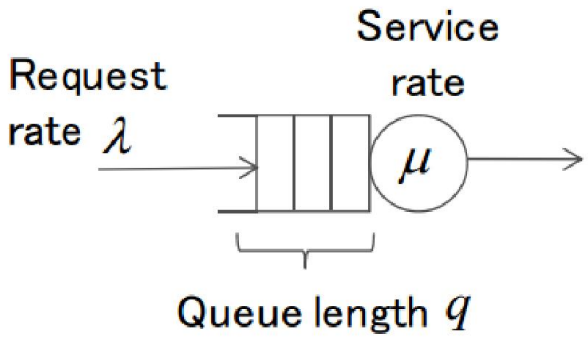


Figure 2.6 Example of a queuing system [27]

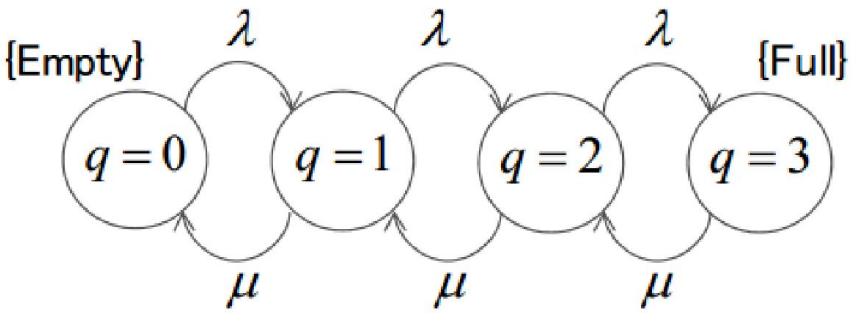


Figure 2.7 CTMC model for the queuing system [27]

Using the PRISM verification tool, we can verify various types of probabilistic properties for this model. For example, we can specify the property "the probability of the queue becoming full within time T is more than 0.1" as "P>0.1 [true U<=T (q=3)]" using CSL language. By conducting verification using the PRISM tool, we can validate that the property is true when T=100.

```
ctmc // Continuous-time Markov chains

const int m      = 3; // Max queue length
const int lambda = 1; // Request arrival rate
const int mu     = 5; // Service rate

module Queue_model

        q: [0..m] init 0; // Queue length

        [Arrival]   (q<m) -> lambda: (q'=q+1);
        [Departure] (q>0) -> mu    : (q'=q-1);

endmodule
```

Figure 2.8 PRISM code for CTMC queuing model [27]

We decided that PRISM is suitable for our performance evaluation of cloud computing on the following reasons. First, PRISM can evaluate the quantitative characteristics of system behaviors such as the probability of violation of service level objectives by state space exploration. This kind of characteristic is suitable for the realm of system management, because this type of information can help system administrators evaluate the degree of credibility and reliability for the system management operations they are going to execute. While some simulation approaches such as NS-2 might also be able to output certain quantitative simulation results, their output is just one execution instance of many possible results

Conclusion

model checking is a model of the system under consideration . this chapter, we surveyed the model checking from many aspects, introduction to system verification , discuss about probabilistic model checking and the different temporal logic. Finally describe the environment PRISM model checker .

Chapter III
PERFORMANCE MODELLING CLOUD COMPUTING SERVICES

1. INTRODUCTION

Cloud computing resources must be compatible, high performance and powerful. High performance is one of the cloud advantages which must be satisfactory for each service. Higher performance of services and anything related to cloud have influence on users and service providers. Hence, performance evaluation for cloud providers and users is important, We summarized the following work from the survey [14]

Nowadays , the term “performance” is more than a classic concept and includes more extensive concepts such as reliability, energy efficiency, scalability and soon. Due to the extent of cloud computing environments and the large number of enterprises and normal users who are using cloud environment, many factors can affect the performance of cloud computing and its resources. Some of the important factors considered are as follows:

- Usability
- Scalability
- Workload
- repetition or redundancy
- Processor Power
- Latency

Cloud computing has attracted considerable research attention, but only a small portion of the work done so far has addressed performance issues, and the rigorous analytical approach has been adopted by only a handful among these. modeled a cloud center as the classic open network, from which the distribution of response time is obtained.

assuming that both inter-arrival and service times are exponential. Using the distribution of response time, the relationship among the maximal number of tasks, the minimal service resources and the highest level of services was found

Yang et al. [13] modeled the cloud center as an M=M=m=m + r queuing system from which the distribution of response time was determined. Inter-arrival and service times were both assumed to be exponentially distributed, and the system had a finite buffer of size m + r. The response time was broken down into waiting, service, and execution periods, assuming that all three periods are independent (which is unrealistic, according to authors' own argument).

Where the authors studied the response time in terms of various metrics, such as the overhead of acquiring and realizing the virtual computing resources, and other virtualization and network communication overhead. To address these issues, they have designed and implemented C-Meter, a portable, extensible, and easy-to-use framework for generating and submitting test workloads to computing clouds. [14]

2. Measurement-Based Evaluation on Cloud Service Performance:

2.1. Representative Work on Measurement-Based Cloud Performance Evaluation :

A general procedure for measurement-based service performance evaluation on a Cloud testbed comprises of the following steps. First the researchers need to specify the purpose and scope of the evaluation, and then identify the features/aspects of the Cloud services that are to be evaluated. The next step is to determine the performance metrics that will be analyzed and select appropriate benchmarks applications for testing. Then an experimental environment should be setup and testing experiments can be performed.

Stantchev presented a general approach for evaluating nonfunctional QoS properties of individual Cloud services in Performance evaluation of cloud computing offerings. Various performance metrics may be used for evaluating different features of Cloud services. A list of

typical performance metrics for evaluating general Cloud services is given in Table 1. A catalogue of metrics for evaluating performance of commercial Cloud services was published in[15], which categories metrics into groups respectively for evaluating the communication, computation, memory, and storage aspects of a Cloud computing platform. Generating appropriate test workload is also an important aspect of measurement-based evaluation methods. A framework for generating and submitting test workloads to evaluating Cloud infrastructure performance was designed in [16].

Metrics	Description
S ervice response time (delay)	the latency time between service request and service completion
Service throughput	the number of jobs that can be processed by the service provider in a time unit
Service availability	the probability that a service request can be accepted by the service provider
System utilization	the percentage of system resources that are busy for service provisioning
System resilience	the stability of system performance over time especially under bursty loads
System scalability	the ability of a system to performance well when it is changed in size or volume
System elasticity	the ability of a system to adapt to changes in its loads

Table 2 Typical metrics used for evaluating Cloud service performance. [14]

In 2006, Amazon Web Services (AWS), a subsidiary of Amazon.com , began offering three web services that allow organizations and individuals to use Amazon's enterprise-class computing infrastructure on an as needed basis and at commodity prices. Amazon's Elastic Compute Cloud (EC2) rents Linux virtual machines at 10 cents per CPU hour; users may rent dozens, hundreds, or even thousands of CPUs simultaneously. Amazon's Simple Storage Service (S3) allows users to store data at a cost of 15 cents per gigabyte per month, with bandwidth priced at between 10 and 18 cents per gigabyte transferred outside Amazon's network. Amazon's Simple Queue Service (SQS) is a reliable messaging service, making it relatively straightforward to coordinate a cluster of computers on a parallelizable large-scale problem, at a cost of 10 cents

per thousand messages. All of these services are sold without startup fees and with no minimum pricing, potentially making them attractive for individuals, universities and corporations alike [17].

Cloud services may achieve quite different levels of performance under various workloads generated by diverse applications. Unlike computation and communication-intensive applications, data-intensive applications typically show strong demands for high-performance I/O and storage access in a Cloud infrastructure.

Evaluation of the performance of Cloud computing platform services may need to handle some special issues, including performance metrics and benchmarks appropriate for PaaS. To address such needs, Atas and Gungor [18] developed a framework for evaluating PaaS performance and proposed a set of benchmark algorithms that can help determine the most appropriate PaaS provider based on different resource needs and application requirements. Commercial PaaS services such as Cloud Foundry, Heroku, and OpenShift, were tested and the obtained results were analyzed by the authors using two evaluation methods: the Analytical Hierarchy Process (AHP) and Logic Scoring of Preference (LSP).

The infrastructure is configured with the Nimbus toolkit to enable remote releasing of resources in a similar manner as EC2 services. OpenCirrus [19] is a large scale Cloud testbed comprising of federated heterogeneous distributed data centers. It enables researchers to exchange data sets and develop standard Cloud computing benchmarks. Virtual machine management in OpenCirrus can be done by different services as long as they are compatible with the EC2 interface. Open Cloud [20] is a research Cloud testbed that is designed to support computations that span more than one data centers. Data centers in Open Cloud are interconnected with a dedicated high-speed wide area network [21].

2.2. Challenges and Opportunities

Measurement-based Cloud service performance evaluations using public commodity Cloud infrastructures as testbeds. The representative works reviewed in last section covered performance of the most popular Cloud services, for example Amazon IaaS

clouds, therefore offer useful insights and practical guidelines to both service providers and consumers regarding service performance evaluation. On the other hand, testing experiments conducted using public Cloud infrastructures as testbeds are facing some challenges, which also offer opportunities for future research.

First, such evaluations can only be performed using the currently avail- able Cloud services with the configuration setting made by service providers. The SOA principle in Cloud provisioning makes internal implementations of Cloud infrastructures transparent to service consumers. Although such transparency greatly enhances the usability and flexibility of Cloud services, it makes measurement-based performance evaluations more difficult.

In most of the experiments reported in the literature, researchers had no control on configuration of the commercial Clouds they used as testbeds. Therefore, whether the results obtained from these experiments with certain settings are applicable to more general scenarios of Cloud service provisioning needs to be further checked.

Secondly, lack of knowledge and control of infrastructure configuration for service deployments limits the researchers' ability to study the impact of resource management inside Cloud infrastructures on service performance through measurement-based evaluations. Service users often want to evaluate the achievable performance of new services to support their decision making on service selection. However, it is difficulty for a service customer to use a measurement-based method to obtain insights about performance behaviors of new Cloud services (or service options) until the customer accepts the service offer and starts using the service.

General measurement-based methods, using either commercial Clouds or research Clouds as testbeds, face some common challenges that offer opportunities for future research. Such methods require extensive and of- ten expensive experimentation and measurements, which must be carefully designed in order to obtain useful results. Although the classification of measurement-based evaluations provided in [46] may serve as a guideline for this purpose, multiple factors including heterogeneity in Cloud implementations, variability in application scenarios, and diversity in users' applications and requirements,

etc., still make designing appropriate experiments for performance measurement a challenging problem for future research.

Selection of testing benchmarks plays a key role in measurement-based performance evaluation for Cloud services. Although traditional bench- marks have been recognized as insufficient for evaluating Cloud services , they are still predominately used in current evaluation research.

Providers offer different services with various capabilities and performance guarantees. On the other hand, research works on performance measurements, even for the same Cloud service, may be conducted in different experimental scenarios. Therefore, another challenge to measurement-based evaluation methods that deserve future investigation is to make the reported results comparable. Developing a standard benchmark for Cloud service testing might help to address this issue.[14]

3. Analytical Modeling-Based Performance Evaluation for Cloud Services

Analytical modeling and analysis techniques have been applied in performance evaluation for Cloud services. Such techniques offer less expensive approaches to analyzing Cloud service performance because they save the cost of testbed experiments required by measurement-based methods.

The queueing theory, as a classical approach for computer system mod- eling and analysis, has been widely employed for evaluating Cloud service performance. Network calculus, which can be viewed as an extension of traditional queueing theory with mini-plus algebra, has also offered a promising profile-based approach for addressing the challenges to performance evaluation brought in by some special features of Cloud computing. Stochastic Reward Net (SRN), an augmentation of Stochastic Petri Net (SPN), has also been exploited by researchers for modeling Cloud service provisioning and analyzing Cloud service performance.

3.1. Performance Modeling Based on Queuing Theory

In order to evaluate Cloud service performance guaranteed to applications with different priorities, developed an M/M/m/m queueing model for Cloud computing centers with multiple priority classes. There are m servers in the model and the total system capacity is m (i.e., no buffer before servers). All servers are split into two categories: reserved servers that are assigned to process client requests by following priority scheduling; and shared servers that are used to serve the request from any client based on a FIFO policy. The model assumes that the service request arrival process to be a Poisson process and that the service time is exponentially distributed. The rejection probability of clients in each priority class was studied as the main performance metric in this study.[22]

In order to represent the bursty arrival workloads of Cloud infrastruc- tures in service performance evaluation. Khazaei et al. modeled Cloud computing centers as an $M^{[x]}/G/m/m+r$ queuing system . In such a model, the arrival service request process is assumed to be a sequence of super-tasks, each of which consists of a burst of tasks. The inter-arrival time of super-tasks is exponentially distributed and the service time of each task in a super-task has a general distribution. The system has m servers and a buffer size of r. A super-task will be rejected if there is no sufficient resource for the whole super-task.[23]

Quantifying Cloud service performance requires appropriate models that can cover vast parameter space. A monolithic model may suffer from in- tractability and poor scalability due to the large number of parameters. An approach to reducing complexity of Cloud service performance analysis is to divide the system model into sub-models and then obtain the overall solution by iteration over individual sub-model solutions.

Author talked Cloud service provisioning is considered to have three main steps: resource provisioning decision, VM provisioning, and run-time execution. Compared to a single one-level monolithic model, this analysis method is more tractable and scalable. Performance analysis based on this model is only applicable to service requests with a single task. However, Cloud users may ask for multiple VMs to handle multi-tasks by submitting a single service request.[24]

the authors employed the idea of sub-model based analysis approach but particularly considered some important features of Cloud computing centers, including batch arrival of tasks and resource virtualization. The authors developed sub-models for resource allocation and virtual ma- chine provisioning and then implemented the sub-models using interactive Continuous Time Markov Chain (CTMC). The sub-models are interactive such that output of one sub-model is the input of the other one and vice versa. The overall solutions for performance metrics such as task blocking probability and total waiting time incurred on user requests were obtained by iteration over individual sub-model solutions.[25]

the authors combined the system details, such as Physical Ma- chine (PM) occupation/release, PM warm-up/cool-down, PM fail/recover, and job rejection, into a general model in order to consider all service provisioning details having simultaneous impacts in determining the overall QoS.[26]

This model captures system behaviors in a general state transition model and considers the inter-influence of these behaviors in deciding the final QoS. Expected service completion time and rejection probability are the QoS metrics this chapter focused on. However, the complexity of the model, mainly due to the state explosion issue of the Marchov chains employed in this thesis , limits the developed techniques only to effectively evaluate small scale Cloud infrastructures. Comparing this combined model against the sub-model methods proposed and indicates the need of a tradeoff between accuracy and complexity in developing analytical models for Cloud service performance evaluation.

3.2. Profile-Based Evaluation of Cloud Service Performance

Queueing techniques for Cloud service modeling and performance analysis are typically developed for specific system architecture with certain assumptions about the service implementations. However, virtualization as the key technology for Cloud computing makes services transparent to their implementations.

Profile-based evaluation methods employ the service curve concept from network calculus to obtain a general profile of service capability that is agonistic to service

implementations. Similarly the arrival curve-based demand profile is able to describe workloads generated by any applications. There- fore, the profile-based performance analysis offers a promising approach to dealing with the heterogeneity in service implementations and diversity in applications in Cloud computing environments. In addition, the profile-based analysis method may also naturally support Cloud-federation and network-Cloud composition.

3.3. Evaluation of Cloud Service Availability

The aforementioned research work about analytical modeling and analysis on cloud service performance focus on delay and throughput-related metrics. That is, the main evaluated performance considered is about how fast a Cloud system may response to a service request and complete the re- quested service, and how many service requests can be handled by a Cloud system in each second. In addition to such delay/throughput-related performance, service availability is another crucial aspect of QoS that must be guaranteed by various Cloud services for meeting user requirements; there- fore, analysis on service availability is also considered as an important part of Cloud performance evaluation.

Availability of Cloud services may be quantified using the rejection prob- ability for a service request; that is, the probability that the request for a service cannot be accepted by the Cloud service provider. The rejection probability is related to multiple factors such as the total system capacities (including both server capacity, buffer space, network bandwidth etc.), Cloud service management mechanisms (e.g., the job scheduling policies employed by the Cloud data center), and characteristics of traffic load arrival at the Cloud system.

4. Challenges and Opportunities

Special features of the Cloud computing paradigm bring in new challenges to system modeling and service performance analysis. In order to accurately represent a Cloud system, an analytical model needs to be scalable to deal with the large amount of resources in Cloud infrastructures and be flexible to handle different implementations and configurations of Cloud services.

Diversity in Cloud service functions, heterogeneity in Cloud implementations, and resource virtualization in Cloud service provisioning are some particular challenging issues that must be fully considered when developing analytical approaches to evaluating Cloud service performance.

All Cloud services are delivered to their end users through networks. Public commercial Cloud services are typically accessed by customers via the Internet and private/hybrid Cloud services need to be accessed through enterprise networks. Therefore, what end users actually perceive is the performance of an end-to-end service offered by both the Cloud infrastructure and the network that provides service access. Therefore, performance analysis from an end user's perspective should consider the combined performance of the Cloud service and the network through which the user accesses the Cloud service. In addition, with the rapid development of Cloud federation, the end-to-end service provisioned to an end user will become a composite service comprising multiple Cloud services and multiple network services.

With rapid development of Cloud federation and the emergence of network- Cloud service convergence, end-to-end service provisioning in the future Cloud environments will often traverse multiple heterogeneous networking and computing domains. The currently available profile-based models for Cloud service performance analysis, although considered network and Cloud service composition, are still limited to scenarios with only a single Cloud infrastructure. Further development of the profile-based evaluation approach in order to fully support Cloud federation offers an interesting topic for future research.

	Advantages	Disadvantages
Measurement-Based Methods	provide performance insights about available Cloud services; reflect service performance in realistic operation scenarios; indicate service performance for practical applications	limited to available Cloud testbeds and their settings; expensive for conducting testing experiments; cannot predict performance for new services and/or settings
Queuing Theory-based Methods(may be applied with SRN)	no experiment cost; can predict performance of new services and settings; may evaluate impacts of a large set of parameters	hard to model heterogeneous Cloud infrastructures; hard to model traffic loads of diverse applications; may not reflect performance of realistic service scenarios
Profile -Based Methods	agnostic to heterogenous Cloud service implementations ; applicable to diverse application loads; reflect virtualization and abstraction features	effectiveness relies on precision of service and demand proles; current analysis only for worst-case performance

Table 3 Advantages and disadvantages of typical types of approaches for evaluating CSP [14]

5. Conclusion

Cloud services have become indispensable ingredients of the future information infrastructures. Evaluation of Cloud service performance is crucial and beneficial to both service providers and service consumers. in this chapter we summarized the related work from a surveys , and we see the currently available approaches to evaluating Cloud service performance are classified based on their research methodologies into two categories: measurement-based approaches and analytical modeling-based approaches.

Chapter IV

ANALYZING PERFORMANCE USING MODEL CHECKER

1. INTRODUCTION

Our work presented here is based on the paper of Khazaei et al. [25] that presented a performance model of Cloud Computing Centers and analyzed this model with respect to various metrics. Basically, this thesis developed and evaluated tractable functional sub-models and their interaction model while iteratively solving them. They constructed separate sub-models for different service steps in a complex cloud center, and the overall solution will be obtained through interaction over the individual sub-models.

In this chapter, we are making one step forward by applying PRISM to performance evaluation of cloud computing based on the model proposed by Khazaei et al. The model is based on constructing separate sub-models for internal and external servicing of a complex cloud center. By introducing a new component to the analytical model, a Pool Management Module (PMM), the performance model supports inter-pool communication. PMs can be moved among pools in order to maintain acceptable availability, response time and power consumption at the same time. More specifically, PMM describes the dynamics of server pools and provides the steady-state pools arrangement. Among problems

The aim behind such model is to help cloud services providers to have a deep insight on the performance of their cloud centers, through having an accurate estimation of performance metrics, in way they could avoid Service Level Agreement (SLA) violations. However, for the model to be used in practical estimation, it must be modelled in such language that can express the CTMC models and their interaction, moreover it must give the possibility to make detailed analysis, especially the quantitative one to estimate the performance metrics in automatic way.

In this chapter, we have constructed for each sub-model described in the paper, a formal model in the language of PRISM . This language essentially allows to construct in a modular manner a finite state transition system and to associate rates to the individual state transitions . The mathematical core of such a system is a Continuous Time Markov Chain

(CTMC) which can be analyzed by the PRISM tool with respect to properties that are expressed in Continuous Stochastic Logic (CSL).

2. Performance Model of a cloud computing center :analytical model :

The article [25] describes the model of a " Cloud Computing Centers" to which the user requests in the queue. Referring to an illustration in Figure 3.1, the authors describe their model as follows:

In IaaS cloud, when a request is processed, a prebuilt or customized disk image is used to create one or more VM instances. In this paper, they assume that prebuilt images fulfill all user requests. We assume that PMs are categorized into three server pools: hot (i.e., with running VMs), warm (i.e., turned on but without running VM).

The steps incurred in servicing a super-task are shown in Fig 3.1 . User requests (supertasks) are submitted to a global finite queue and then processed on a first-in, first-out basis (FIFO).

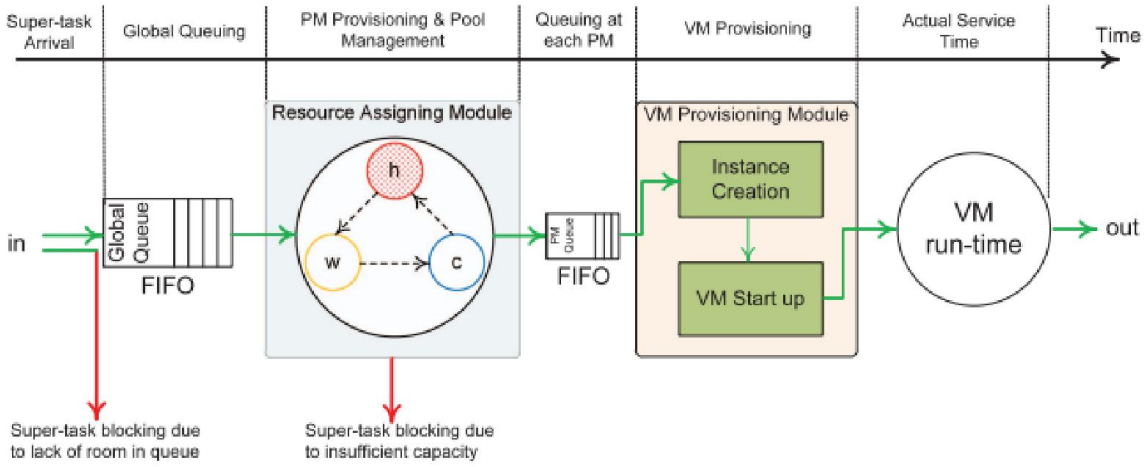


Fig 4.1:. The steps of servicing and corresponding delays.[25]

The model proposed allows users to request more than one VM by submitting a super-task . λ_{st} a super-task may contain more than one task, each of which requires one VM. The size of super-task is assumed to be geometrically distributed.

The complete list of parameters defining the performance of the model (with default values stated in parameters) is given as

Symbol	Description
λ_{st}	Arrival rate of super-tasks
Lq	Size of global queue "Buffer Size
MSS	Maximum Super-task Size
λ_h, λ_w, λ_c	Arrival rate to a PM in hot, warm and cold pool
RAM	Resource Assigning Module
RASM	Resource Allocation Sub-Mode
VMPSM_x	Virtual machine provisioning submodel for each pool

Table : Symbols and Corresponding Descriptions

The first analytical model, the Resource Management Module, captures the details of queuing at the global admission control and at the PMs input queue. The resource assignment process is modeled with a two dimensional continuous time Markov chain shown in Fig 3.1

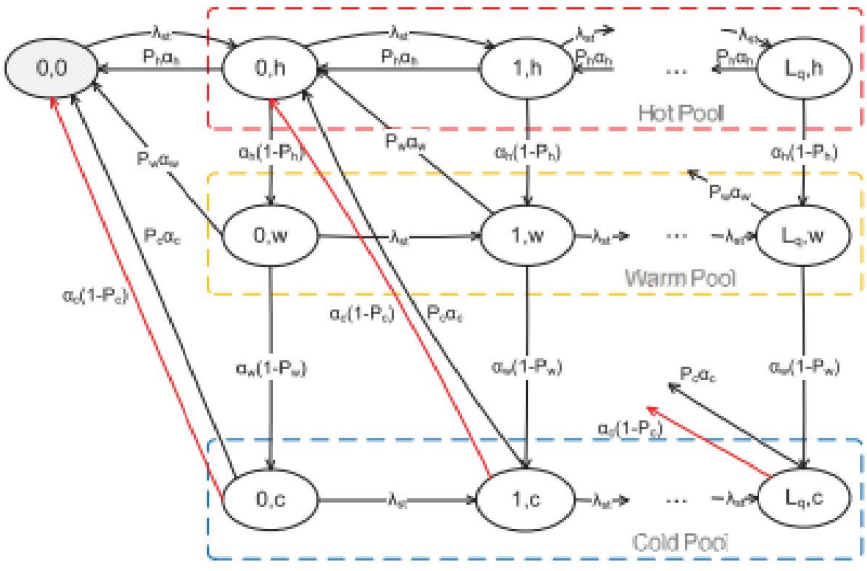


Fig 4.2:Resource allocation submode .**[25]**

Since the number of potential users is high and a single user typically submits super-tasks at a time with low probability, the super-task arrival can be adequately with rate λ_{st}. super-tasks are lined up in the global finite queue to be processed in a first-in, first-out (FIFO) basis. Each state of Markov chain is labeled as (i, j), where i , indicates the number of STs in queue and j denotes the pool on which the leading ST is under provisioning. State (0;0) indicates that system is empty, which means that there is no request under provisioning or in the queue. Index j can be h; w,or c that indicate current ST is undergoing provisioning on hot, warm, or cold pool, respectively. The global queue size is Lq and one more ST can be at the deployment unit for provisioning, so the total system capacity will be Lq +1.

The rejection at (x,h) states is due to insufficient resources at the moment and rejection at states (Lq ,h) and (Lq ,w) is due to lack of space in the global queue.

The second analytical model is Virtual Machine Provisioning Sub-Model (VMPSM) that captures the instantiation, deployment and provisioning of VMs on a PM. the VMPSM(an approx-imated-CTMC) for a PM in hot pool. A PM in warm or cold pool can be modeled with the same VMPSM, though, with different arrival and instantiation rate. Consequently, each pool (hot, warm and cold) can be modeled as a set of VMPSM with the same arrival and instantiation rate.

We assume that the requests of STs arrive according to a Poisson process with rate λ_{st}, Let st be the average of the requested file size i. We define λ_h, λ_W and λ_c such that:

- $\lambda_h = \lambda_{st} * (1 - Bpq) / N_h$
- $\lambda_w = \lambda_{st} * (1 - Bpq)(1-P_h)/N_w$
- $\lambda c = \lambda_{st} * (1 - Bpq)(1-P_h) *(1-P_w)/N_c$

In our model we assume that the RASM has a buffer of capacity lq. Let Px (x=hot,warm,cold) be the probability that a request will be denied by the RASM. As it is well known from basic queueing theory the blocking probability Bpq for queueing system.

Now we can see that the requests arrive to the buffer of the RASM according to a Poisson process with rate λst. If the size of the requested file is greater then the RASM output buffer it will start a looping process until the delivery of all requested file's is completed. Let q = st/t be the probability that the desired file can be delivered at the first attempt. Let λst be the rate of

the requests arriving at the hot pool considering the looping process. According to the conditions of equilibrium and the flow balance theory of queueing $\lambda = q * \lambda st$.

The ST under provisioning decision leaves the deployment unit, once it receives a decision from RASM and the next ST at the head of global queue will go under provisioning. In this sub-model, arrival rate of ST (λst) and look-up delays (1/ah;1/aw,and1/ac) are exogenous parameters and success probabilities (Ph,Pw,and Pc) are calculated from the VM provisioning sub-model.

Using steady-state probabilities some performance metrics such as blocking probability and probability of immediate service can be calculated. Two types of blocking may happen to a given ST:

Blocking due to a full global queue occurs with the probability of:

$BPq = \pi_{(lq.h)} + \pi_{(lq.w)} + \pi_{(lq.c)}$

Applying Little's law , the mean waiting time in queue(wt) is given by:

$Wt = q/(\lambda st(1-Preject))$

3. PRISM Implementation

We have seen that the model has two new issues: (1) external visits(arrival STs), (2) the RASM has limited buffer. In PRISM any queue must have a buffer limit. This means that we have to deal only with the first issue, i.e., we have to simulate external STs. We simulate external users by the following new module:

```
module submit1

 [Arrival_i] (st <lq) true -> lambda: true;

 endmodule
```

This module generates external requests with rate λ, which is denoted in the PRISM code by lambda. The requests are sent to the buffer , i.e., we have to synchronize with the hot pool, therefore, the corresponding transactions have the same label, which is "arrival_i". We show only those lines from the rasm module, which are not included in the base implementation:

```
module rasm

 [Arrival_i] (i<lq) &(h=0)-> (i'=i+1)&(h'=1);
```

```
[Arrival_i] (i<lq) &(h=1)-> lamh:(i'=j)&(h'=0) & (w'=1);

[exec_i] (i>0 )&(i<lq)&(h=1) -> muPhah :(i'=i-1)&(h'=h);

Endmodule
```

After an external request has arrived, it is processed and the answer is placed in the output queue, which is simulated by the vmpsm module that will distribute the request to each pool (hot ,warm ,cold pool).

So we do not know which answer belongs to which kind of answers, but we know the incoming rate of the two kind of requests:

We know a PM must be at least equal to MSS ,vmpsm requests arrive with rate $\lambda_h = \lambda_{st} * (1 - Bpq) / N_h$ And $\lambda_w = \lambda_{st} * (1 - Bpq)(1-P_h)/N_w$, , $\lambda c = \lambda_{st} * (1 - Bpq)(1-P_h) *(1-P_w)/N_c$. So we know that N is the number of PM in each pool, λh/λw/λc/ is the arrival rate of hot, warm, cold.

We used this observation in the `vmpsm_hot` module. We show only those lines from the `vmpsm_hot` module, which are altered or not included in the base implementation:

```
module vmpsm_hot

...

          [Arrival_x] (x<mss) -> lamh :(x'=x+1);

          [Arrival_x] (x<mss) &(h=1)-> lamfi:(x'=y)&(h'=0) & (w'=1);

          [exec_x] (x>0 )&(h=1) -> mu:(x'=x-1)&(h'=0);

Endmodule
```

Finally we had to rewrite the timing rewards.

- The original time reward was :
```
rewards "waiting"
        i>=1 & !(i>=1 & h=1) : 1;
        endrewards.
```

- Reward when the served in hot pool :
```
rewards "served_hot"
        [Arrival_x] true :1;
        endrewards
```

Reward when

❖ Reward when the served in warm pool rewards "served_warm" [Arrival_y] true : 1;

endrewards

❖ Reward when the served in cold pool rewards "served_cold" [Arrival_z] true : 1;

endrewards

❖ The total model code as following a.1:

```
ctmc
// variable parameters
const double lambda= 6;  //  arrival rate of super task
const double p;  // probability of having st with size i
//////////////////////
const lamfi=1 ;
const mu =4;
const mss=5 ;//max super-task size
//const double p;
// parameters from paper (units are bytes and seconds)
const double st =5; //super task
const  lq =2;            //global queu  // buffer size
//  look up rates
const  double ah=625;
const  double aw=825;
const  double ac=825;
const double wt =q/(lambda *(1-Prej));
const double lut =((1/ah)+(1-ph)*((1/aw)+(1-pw)*(1/ac)))/1-
const q ;
// probabilities
const double ph=0.5 ;       // prob. off succes in hot
const double pw =0.3 ;       //  prob. off succes in warm
const double pc=0.3 ;       //  prob. off succes in cold
//arrival rate  to pm
const double lamh = ah* (1-ph) ;
const double lamw=aw * (1-pw);
const double lamc =ac* (1-pc);
const double muPhah=ph*ah;
const double muPwaw=pw*aw;
const double muPcac=pc*ac;
// helper constans
const double Nh=15 ;   //Number of PMs in hot
const double Nw =20;   //Number of PMs in  warm
```

```
34 const double Nc=15 ; // Number of PMs in cold
35 const double Bpq=0.2; // prob .of blocking to lack of room  in global queu
36 const double Bpr=0.2 ; // prob .of blocking to lack of capacity
37 const double Prej=Bpq+Bpr;//total prob of blocking
38 const double lambdah = lambda*(1-Bpq)/Nh;
39 const double lambdaw =(lambda*(1-Bpq)*(1-ph))/Nw;
40 const double lambdac =(lambda*(1-Bpq)*(1-ph)*(1-pw))/Nc;
41 module Submit1 // Request arrivals for rasm
42  [Arrival_st] (i +j+k <lq) -> lambda: true;
43 endmodule
44 // Modules for request arrivals
45 module rasm
46         i: [0..lq];
47         h: [0..1];
48         j: [0..lq];
49         w: [0..1];
50         k: [0..lq];
51         c: [0..1];
52         [Arrival_st] (i<lq) &(h=0)-> lambda :(i'=i+1)&(h'=1);
53         [Arrival_st] (i<lq) &(h=1)-> lamh :(i'=j)&(h'=0) & (w'=1);
54         [exec_st] (i>0 )&(i<lq)&(h=1) -> muPhah :(i'=i-1)&(h'=h);
55         [Arrival_st] (j<lq)&(w=0) -> lambda: (j'=j+1)&(w'=1);
56         [Arrival_st] (j<lq)&(w=1) -> lamw:(j'=k)&(w'=0) & (c'=1);
57         [exec_st] (j>0 )&(j<lq)&(w=1) -> muPwaw :(j'=j-1)&(w'=h);
58         [Arrival_st] (k<lq)&(c=0) -> lambda :(k'=k+1)&(c'=1);
59         [Arrival_st] (k<lq) &(c=1)-> lamc:(k'=i)&(c'=h);
60         [exec_st] (k>0 )&(k<lq)&(c=1)-> muPcac :(k'=k-1)&(c'=0) & (h'=1);
61   endmodule
62 module Submit2 // Request arrivals for vm
63  [Arrival_x] (x+hh <mss) -> lambdah: true;
64 endmodule
65 module vmpsm_hot
66 x: [0..mss] init 0;
```

```
hh: [0..1] ;
                [Arrival_x] (x<mss) -> lambdah :(x'=x+1);
                [Arrival_x] (x<mss) &(hh=1)-> lamfi:(x'=y)& (hh'=0);
                [exec_x] (x>0 )&(x<mss)&(hh=1) -> mu:(x'=x-1)&(hh'=0);
endmodule
module Submit3 // Request arrivals for vm
 [Arrival_y] (y+ww <mss) -> lambdaw: true;
endmodule
module vmpsm_warm
y: [0..mss] init 0;
ww: [0..1] ;
                [Arrival_y] (y<mss)&(ww=0) -> lambdaw:(y'=y+1)&(ww'=1);
                [Arrival_y] (y<mss)&(ww=1) -> lamfi:(y'=z)&(ww'=0) ;
                [exec_y] (y>0 )&(y<mss)&(ww=1) -> mu :(y'=y-1)&(ww'=h);
endmodule
module Submit4 // Request arrivals for vm
 [Arrival_z] (z+cc <mss) -> lambdac: true;
endmodule
module vmpsm_cold
z: [0..mss] init 0;
cc: [0..1] ;
        [Arrival_z] (z<mss)&(cc=0) -> lambdac:(z'=z+1)&(cc'=1);
        [Arrival_z] (z<mss) &(cc=1)-> lamfi :(z'=x)&(cc'=hh);
        [exec_z] (z>0 )&(z<mss)&(cc=1)-> mu :(z'=z-1)&(cc'=0) ;
endmodule
label  "allserved" = (x+y+z =st);
label  "served_hot" = (x >=1 & hh=1);
label  "served_warm" = (y >=1 & ww=1);
label  "served_cold" = (z >=1 & cc=1);
label  "served_hot_Rasm" = (i >=1 & h=1);
label  "served_warm_Rasm" = (j >=1 & w=1);
label  "served_cold_Rasm" = (k >=1 & c=1);
label  "waiting_h" = i>=1 & !(i>=1 & h=1) ;
```

```
// expected time station 1 is waiting to be served
rewards "waiting"
        i>=1 & !(i>=1 & h=1) : 1;
endrewards
rewards "served_hot"//nubre of st servd by each pool
        [Arrival_x] true : 1;
endrewards
rewards "served_warm"
        [Arrival_y] true : 1;
endrewards
rewards "served_cold"
        [Arrival_z] true : 1;
endrewards
rewards "time" //time request spends in queue
true :i/lq ;
endrewards
rewards "time1" //time spent in system
true :(i/lq)+ (i/mss );
endrewards
```

4. Test Results:

The PRISM implementation of this model can be found in Appendix a.1 We are now going to explain the implementations:

Checking the Estimated waiting Time in Figure 4.3 and Figure 4.4 .the properties and parameter using in this simulation as folows :

- R{"waiting"}=? [C<=T]
- lambda= 6;
- p =0.1;
- st =5;
- lq=2;
- q =1

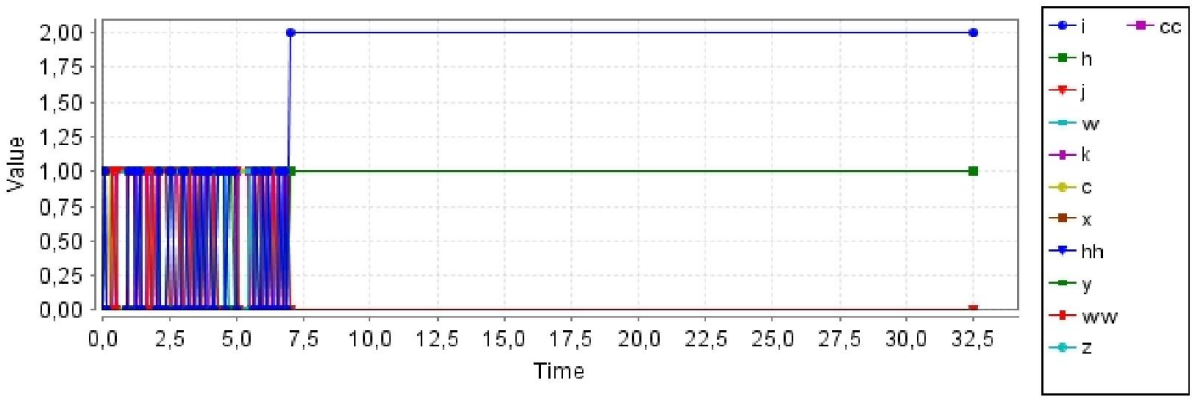


Figure 4.3 Estimated waiting Time b

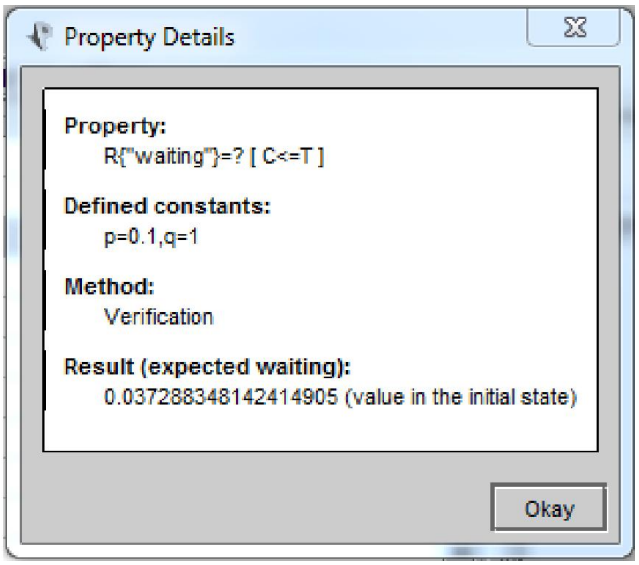


Figure 4.4 Property.1 waiting Time

The second one is the Estimated response Time of the new implementation in case lambda =5. We can see that the diagrams are virtually almost the same,

- lambda=5;
- p=0.1;
- st =4;
- lq =2;
- q=1
- the properties using in this simulation is :
- R{"waiting"}=? [C<=T]

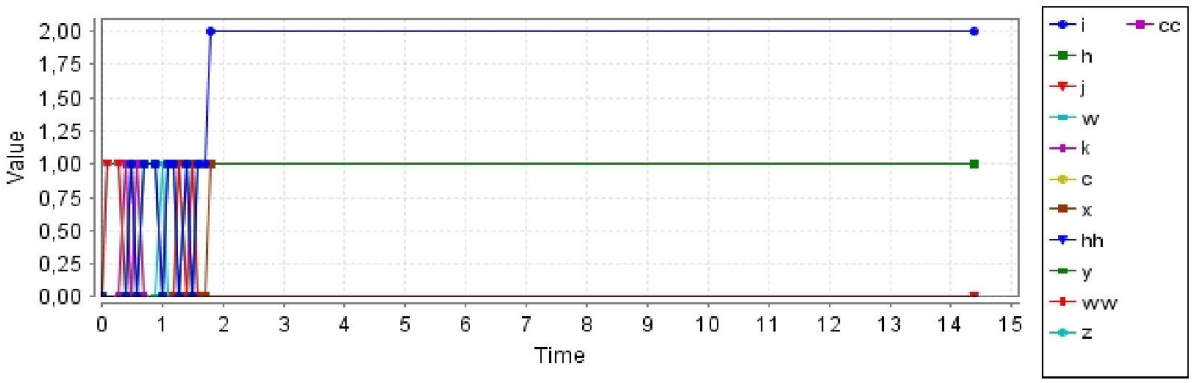


Figure 4.5 the Estimated waiting Time

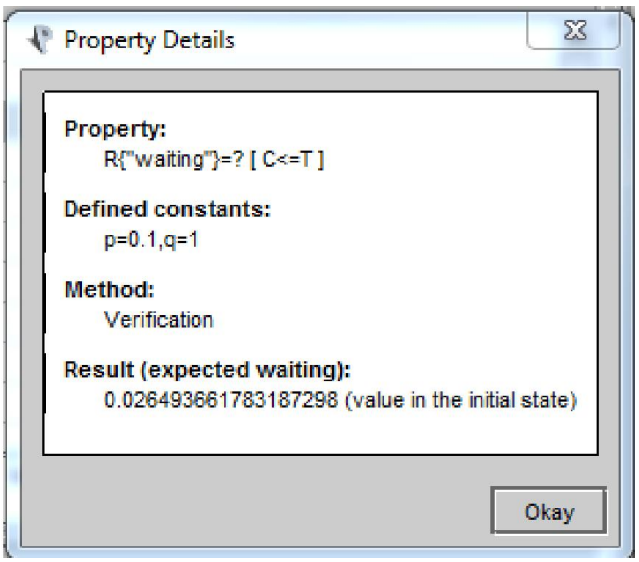


Figure 4.4 Property. 2 waiting Time

In Figure 4.5 we can see the Estimated waiting Time of the new implementation using the "time" reward and the parameters shown above.

- The Figre 4.6 represent the Probability that station 1 will be polled within T time units the parameter and properties using in this simulation is :
- P=? [true U<=T "allserved"]
- lambda=5;
- p=0.1;
- st =4;
- lq =2;
- q=1

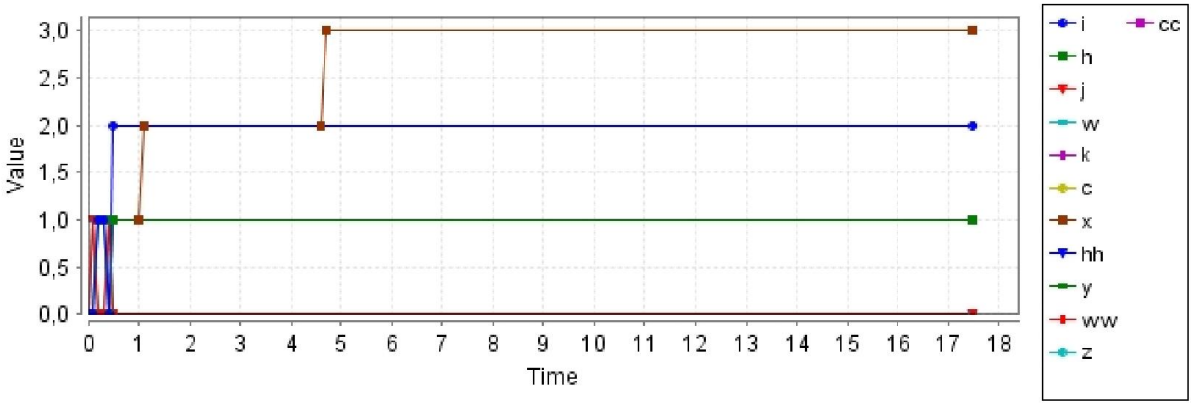


Figure 4.5 Probability that station 1 will be polled within T time units

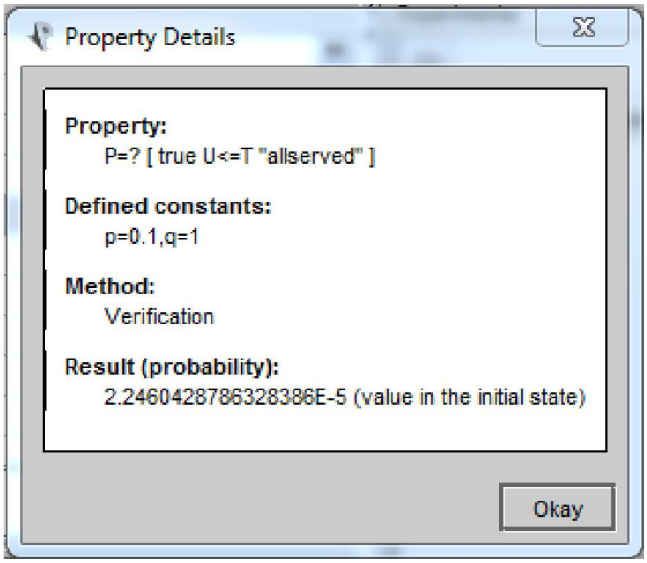


Figure 4.6 Property of Probability that station 1 will be polled within T time units

- Now we see Probability that in the long run station 1 is a waiting service in Figure 4.7 the properties and parameter using in this simulation is :
- S=? [i=1&!(j=1&w=1)]
- lambda=5;
- p=0.1;
- st =4;

- lq =2;
- q=1

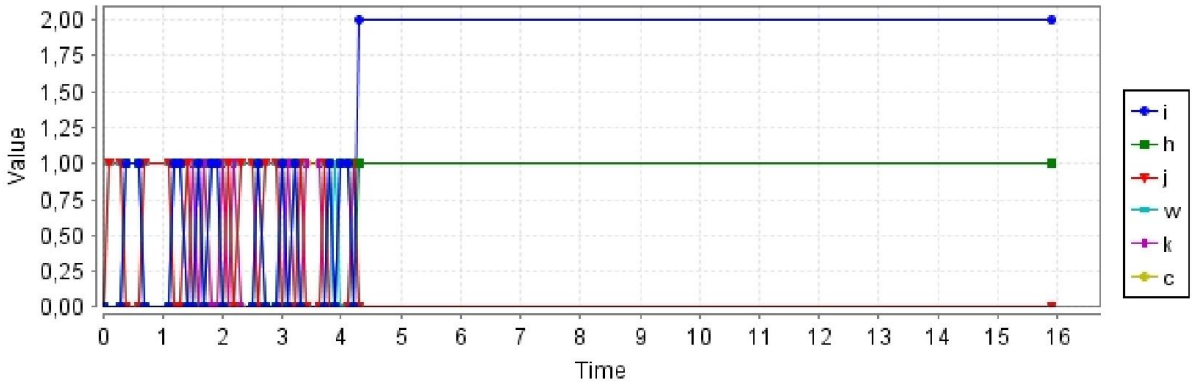


Figure 4.7 waiting service

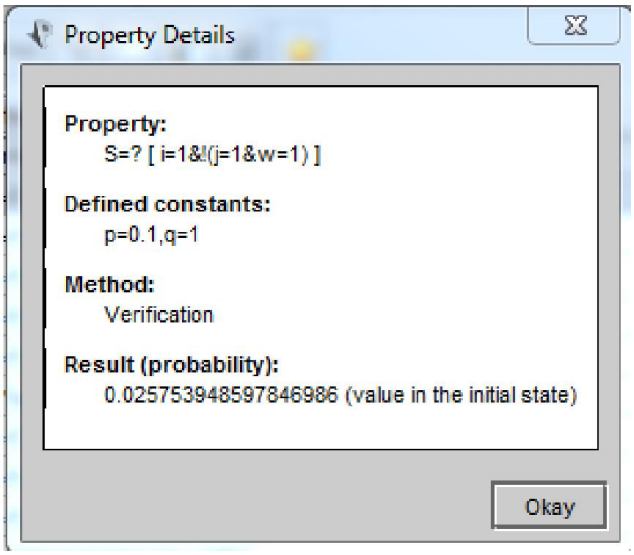


Figure 4.8 Property waiting service

5. Property Specification

- Probability that in the long run station 1 is awaiting service .

S=? [i=1&!(j=1&w=1)]

- Probability that in the long run station 1 is idle

S=? [j=0]

- Once a station becomes full, the minimum probability it will eventually be polled

P=? [true U (i=1&k=0) {j=1}{min}]

- Probability, from the inital state, that station 1 is served before station 2

P=? [!(i=2&k=1) U (i=1&k=1)]

- Probability that station 1 ill be polled within T time units

P=? [true U<=T "allserved"]

- Expected time station 1 is waiting to be served

R{"waiting"}=? [C<=T]

- Expected time when the hot pool is served

R{"served_hot"}=? [C<=T]

- Expected time when the warm pool is served

R{"served_warm"}=? [C<=T]

- Expected time when the cold pool is served

R{"served_cold"}=? [C<=T]

- Expected time in queue

R{"time"}=? [S]

- Expected time in system

R{"time1"}=? [S]

6. Conclusion:

In this chapter we developed the model in Prism model checker and did a simulation work and introduced some parameters in CSL language and showed us the results as shown above.

General conclusion

In this thesis, we were able to accomplish the proposition that we have set in the general introduction. We have developed an interactive analysis model where we started with a basic analytical model and we have extended our analytical model for performance evaluation of highly virtualized cloud computing centers using the PRISM model checker, our work is represented in modeling our problem in PRISM.

This model can help predict the probability of mission rejection, power consumption, and steady-state. We can write the above possibilities in the language of the CSL. We will explain the steps of implementation as follows:

We have developed separate sub-models for different resource allocations (RASM) in the physical machine where we take care of a number of ST in queues and then pass them to any pool and we construct a VM Provisioning Sub-model for each pool (hot, warm and cold).

We have introduced in the model many parameters to study the probability of rejection of the task. The probability of success in hot, warm and cold pool, mean waiting time in global queue. The probability of having a super-task size, super-task rate, task time, and virtualization. And in the last simulation tool via Prism Model Checker.

There are some probabilities mentioned in the article but we could not write or calculate them in our thesis. We could not write because of the inaccuracy of the analytical paper in the form of CSL, where we noted that we cannot rely on what is found in the analytical paper. But in the future, we want to continue work in the analysis modeling of the performance of the cloud and try to achieve all the problems that we did not reach in this thesis.

References

[1] Ms. Shubhangi Ashok Kolte,Prof. P E Ajmire, “A Survey -Cloud Computing”, Internal Journal of Advanced Research in Computer Science and Software Engineering, Vol. 5, Issue 4, April 2016

[2] Virendra Singh Kushwah, Aradhana Saxena, ”.A Security approach for Data Migration in Cloud Computing” ,International Journal of Scientific and Research Publications, Volume 3, Issue 5, May2013

[3] . K. Chandrasekaran “Essentialsof CLOUD COMPUTING” by Chapman and Hall/CRC, December 5, 2014

[4]Cloud Security Alliance. (2009). Security Guidance for Critical Areas of Focus in Cloud

Computing.

[5] Christel Baier and Joost-Pieter Katoen , ‘Principles of Model Checking’, the MIT Press ,May 2008.

[6] Pieter Katoen , Masters Thesis in Computer Science of H.A. Oldenkamp ,”Probabilistic model checking A comparison of tools” , May, 2007

[7] Dhananjay Raju “ LTL and CTL “ACM Conference on Computer and Communications Security (14th 2007:Alexandria, Va.).

[8] Hichem DEBBI, Systems Analysis using Model Checking with Causality, Mar,2015

[9] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert K. Brayton. Verifying continuous time Markov chains. In Rajeev Alur and Thomas A. Henzinger, editors, Proc. 8th International Conference on Computer Aided Verification (CAV’96), Berlin, 1996.

[10] Christel Baier, Joost-Pieter Katoen, and Holger Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In Jos C. M. Baeten and Sjouke Mauw, editors, Proc., Berlin, 1999.

[11] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst.1986.

[12] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Modelchecking algorithms for continuous-time Markov chains. IEEE Transactions on Software Engineering, 2003.

[13]_Hamzeh Khazaei, Jelena V. Misic, Vojislav B. Misic, Nasim Beigi Mohammadi: Availability analysis of cloud computing centers. GLOBECOM 2012

[14] Qiang Duan , Cloud Service Performance Evaluation: Status,Challenges, and Opportunities { A Survey from the System Modeling Perspective ‘December 2016.

[15] Li, Z., O'Brien, L., Zhang, H., Cai, R., On a catalogue of metrics for evaluating commercial cloud services. In: Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing . 2012.

[16] Yigitbasi, N., Iosup, A., Epema, D., Ostermann, S., C-meter:a framework for performance analysis of computing Clouds. In: Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid. 2009.

[17] Garfinkel, S. L., An evaluation of Amazons Grid computing services: EC2, S3, and SQS. Tech. rep., Center for Research on Computation and Society, Harvard University. 2007.

[18] Atas, G., Gungor, V. C., Performance evaluation of cloud computing platforms using statistical methods. Computers & Electrical Engineering . 2014.

[19] Avetisyan, A. I., Campbell, R., Gupta, I., Heath, M. T., Ko, S. Y., Ganger, G. R., Namgoong, H. Open cirrus: A global cloud computing testbed. (2010).

[20] Grossman, R., Gu, Y., Sabala, M., Bennet, C., Seidman, J., Mambratti, J., The open Cloud testbed: a wide area testbed for Cloud computing utilizing high performance network services 2009.

[21]Sakellari, G., Loukas, G.,. A survey of mathematical models, simulation approaches and testbeds used for research in Cloud computing. 2013

[22]Ellens, W., Zivkovic, M., Akkerboom, J., Litijens, R., Berg, H., Performance of Cloud computing centers with multiple priority classes. In: Proceedings of the 2012 IEEE

[23] Khazaei, H., Misic, J., v. B. Misic. Performance analysis of Cloud computing centers under burst arrivals and total reject policy. In: Proceedings of the 2011 IEEE

[24] Ghosh, R., Longo, F., Naik, V. K., Trivedi, K. S., Modeling and performance analysis of large scale IaaS Clouds. Future Generation Computer Systems 2013.

[25] Khazaei, H., Misic, J., v. B. Misic, A fine-grained performance model of Cloud computing centers. IEEE Transactions on Parallel and Distributed System . 2013.

[26] Xia, Y., Zhou, M., Luo, X., Zhu, Q., Li, J., Huang, Y.,. Stochastic modeling and quality evaluation of infrastructure-as-a-service Clouds. IEEE 2015.

[27] shinji kikuchi & yasuhide matsumoto Performance Modeling of Concurrent Live Migration Operations in Cloud Computing Systems using PRISM Probabilistic Model Checker 2011.

[28] site niste, www.niste.org, consulté le : 05/05/2013

ملخص:

مصطلح الحوسبة السحابية يمكن تعريفه بأنه عبارة عن مجموعة من الخدمات التى تقدم من مزود الخدمة إلى عميل أو عدة عملاء عبر الإنترنت . ويعتبر تقييم أداء خدمات الحوسبة السحابية حاسما ومفيدا لكل من مقدمي الخدمات ومستهلكي الخدمات ويؤثر على الأداء العالي للخدمات وأي شيء يتعلق بالسحابة على المستخدمين ومقدمي الخدمات . وبالتالي قمنا بتقييم أداء الحوسبة السحابية عن طريق النمذجة التحليلية. وباستخدام المدقق بريسم .

الكلمات المفتاحية : الحوسبة السحابية، ونمذجة التحليل، وتقييم الأداء

Abstract:

The term cloud computing can be defined as a set of services provided by the service provider to a customer or several clients over the Internet. The performance assessment of cloud computing services is critical and beneficial to both service providers and service consumers. The high performance of services and anything related to Clouds affects users Therefore, we have evaluated the performance of cloud computing by analysis modeling using "PRISM model checker".

key words : cloud computing ,analysis modeling ,evaluated the performance

Résumé:

Le terme cloud computing peut être défini comme un ensemble de services fournis par le fournisseur de services à un client ou plusieurs clients sur Internet. L'évaluation de la performance des services de cloud computing est essentielle et bénéfique tant pour les fournisseurs de services que pour les consommateurs de services. La haute performance des services et tout ce qui concerne les nuages affecte les utilisateurs. Par conséquent, nous avons évalué les performances du cloud computing par modélisation d'analyse à l'aide du modèle PRISM.

Mots clés: cloud computing, modelage d'analyse, évaluation de la performance