USING FORMAL METHODS TO VALIDATE THE USAGE, PROTOCOLS, AND FEASIBILITY IN LARGE SCALE COMPUTING SYSTEMS

A Dissertation
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Saif Ur Rehman Malik

In Partial Fulfillment
for the Degree of
DOCTOR OF PHILOSOPHY

Major Department:
Electrical and Computer Engineering

April 2014

Fargo, North Dakota

North Dakota State University
Graduate School

Title

Using Formal Methods to Validate the Usage, Protocols, and Feasibility in Large Scale Computing Systems

By

Saif Ur Rehman Malik

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

DOCTOR OF PHILOSOPHY

SUPERVISORY COMMITTEE:

Samee U. Khan
Chair

Sudarshan K. Srinivasan

Jacob S. Glower

Ying Huang

Approved:

06/10/2014	Scott C. Smith
Date	Department Chair

ABSTRACT

A paradigm shift has occurred in the Information and Communication Technology sector. The main obstacle to relegate complex and sensitive tasks is not the inadequate speed and unsatisfactory computing power of the existing machines. However, the inability to design and implement the systems, with a desirable degree of confidence in the correctness and reliability, under different circumstances, has crept in to be the primary concerns in achieving high performance. The hardware and software systems are growing inevitably in scale and functionality, such as cloud computing systems and Data Center (DC). In the said perspective, the complexity of the systems is also increasing. The likelihood of elusive errors is directly proportional to the complexity of the systems that also increase the cost of errors while the systems are operational. In large scale systems the density of computational devices is in order of tens of thousands of servers. Moreover, the effects of errors and miscalculations are substantial. Furthermore, if the specified quality of service is not delivered by the cloud service providers, then the reputation may fall down and users will not use the services, resulting in huge financial lose. Therefore, the reliability, robustness, and availability of systems are very essential. In the said perspective, to increase the reliability and correctness of the systems, we propose the use of Formal Methods (FM). The FM use sound mathematical foundations to prove program correctness. The aim of our research is to deploy various FM tools and techniques to formally analyze the behavior and correctness of the strategies, such as routing algorithms and virtualization models that are implemented in large scale computing systems. The goal of our research is to thoroughly study the strategies, highlight the grey areas that can be further exploit to increase the reliability and performance, and propose a feasible solution. The large scale computing systems, specifically DC exhibits different architectural characteristics, such as predefined complex architectural and topological pattern composed in different layers. The aforementioned characteristics of the underlying network along with the large scale of the servers situate several challenges for the adoption of FMs strategies.

ACKNOWLEDGEMENTS

First of all I would like to thank Almighty Allah for giving me the strength, courage, health, and abilities to pursue my PhD and finish the disquisition.

My deepest and sincere gratitude goes to Dr. Samee U. Khan, my senior advisor for everything I have learned from him, and his continuous help and support during all the stages of this disquisition. I would also like to thank him for all the motivation that I needed when I was down. I truly believe that without his guidance, vision, and encouragement the completion of this disquisition was not possible. I thank him for believing me, when I did not had the believe on myself.

I would also like to mention Dr. Sudarshan K. Srinivasan, my co-advisor for all the help, guidance, and suggestions. Special thanks to all the committee members, Dr. Jacob S. Glower and Ying Huang for their support, guidance and helpful recommendations. Their guidance has served me well and I owe them my heartfelt appreciation.

Thanks to the Electrical and Computer Engineering staff members Jeffrey Erickson, Laura D. Dallman, and Priscilla Schlenker for all the unconditional help and favor.

I wish to thank my family, Aijaz Hussain (Father), Anjum Syed (Mother), Fazal Ur Rehman and Mehmood Ur Rehman (Brothers), and my sweet and sour Fatima Aijaz (Sister). Their support and love was the driving force behind me. I owe them a lot and wish I could show them how much I love and appreciate them. My wife, Hirra Saif, whose encouragement and patience allowed me to finish this journey. She already has my heart so I will just acknowledge her by saying a sincere and heartfelt “thanks”. My two little angels Mohid and Ayaan, their smiles and laughter used to loosen me up from all the stress and worries.

Finally I would like to thank my friends, here in US and in Pakistan for all the moments of fun, laughter, and joy.

DEDICATION

I would like to dedicate this thesis to my family, especially to my mother and my wife for all the inexplicable love, support, and motivation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1. Large Scale Computing Systems

As we delve deeper into the ‘Digital Age’, we witness an explosive growth in the volume, velocity, and variety of the data available on the Internet. For example, in 2012 about 2.5 quintillion bytes of data was created on a daily basis that originated from myriad of sources and applications including mobile devices, sensors, individual archives, social networks, Internet of Things, enterprises, cameras, and software logs. Such ‘Data Explosions’ has led to one of the most challenging research issues of the current Information and Communication Technology (ICT) era: how to optimally manage (e.g., store, replicated, filter, and the like) such large amount of data and identify new ways to analyze large amounts of data for unlocking information? It is clear that such large data streams cannot be managed by setting up on-premises enterprise database systems, as it leads to a large up-front cost in buying and administering the hardware and software systems. In the said perspective, the emergence of technological advances, such as multicore processors and networked computing environments, has helped software practitioners to achieve the vision of creating a software paradigm for millions of users to use as a service [1]. The Large Scale Computing Systems (LSCS), such as cloud computing is one such paradigm with which a shared pool of resources (networks, servers, storage, applications, and services) can be accessed conveniently and on-demand. Moreover, the services can be rapidly provisioned or released with minimal management effort or service provider interaction [2].

The LSCS, such as cloud computing, has been a mainstream of research in last few years. In a report [3], the cloud computing is listed as a top research trend from the year 2006 to 2010. In LSCS, substantial data analysis applications are executed that requires massive amount of memory, processor cycles, and communication bandwidth. To perform substantial computations and meet the ever increasing demands of users, the LSCS are equipped with an order of tens of thousands of servers. Amazon, Google, IBM, Facebook, and Microsoft have started to establish Data Centers (DC) that host cloud computing applications in geographically distributed locations [4]. To maintain and deliver the specified Quality of Service (QoS) attributes, such as throughput, the LSCS must operate in a smooth and efficient way all the time. The income of the DC is defined by the Service Level Agreement (SLA), which outlines the amount paid by the users based on the QoS they receive. The computational and operating margins of DCs depend highly on the provision of the QoS. Higher QoS attribute levels lead to higher rates that in turn lead to higher computations.

1.2. Formal Methods

Formal Methods (FM) used in developing computer systems are rigorous mathematically based tools and techniques that describes system properties, and are used for the specification, development, and verification of software and hardware systems. The FM techniques have matured considerably as a verification discipline in the past few decades and have become a mainstream technology in industrial design, verification methodologies, and processes. Moreover, the increasing criticality and complexity of applications along with the role of software and hardware in those applications has led to the maturity of FM techniques. The aim of such techniques is to increase the quality of software by mathematically proving program correctness as opposed to using test cases. The method or technique is considered to be "Formal", if it is backed up by sound mathematical grounds, which is typically provided by the specification language. The mathematical bases are used as a mean to precisely define notions, such as consistency, completeness, specification, implementation, and correctness [5]. The mathematical structure involved in FM also helps in proving that if the system is implemented correctly, then the specification is realizable. The specification provides a complete description of the behavior of a system to be developed and also includes use cases to describe user interactions with the software. In software engineering, specification is the intermediate product of the software development process. Moreover, the correctness of a system or program can be determined using specification.

The FMs are mainly used to reveal incompleteness, ambiguity, and inconsistency in a system. However, it is noteworthy that the use of formal methods does not miraculously guarantee the aforesaid results, but can be used to increase the level of confidence towards the correctness of the system. FMs can be used in different stages of software development life cycle. The use of FMs in the early stages of development process can reveal design flaws that otherwise might be discovered in the costly stages of testing and debugging phases. When used at the later stages, FMs can help in determining the correctness of the systems implementation. Moreover, the quality and reliability of software is increased by FM techniques using rigorous mathematical modeling, analysis, and verification.

1.3. Motivation

The main obstacle to relegate complex and sensitive tasks is not the inadequate speed and unsatisfactory computing power of the existing machines. However, the inability to design and implement the systems, with a desirable degree of confidence in the correctness and reliability, under different circumstances, has crept in to be the primary concerns in achieving high performance. The hardware and software systems are growing inevitably in scale and functionality, such as cloud computing systems. In the said perspective, the complexity of the systems is also increasing. The likelihood of elusive errors is directly proportional to the complexity of the systems that also increase the cost of errors while the systems are operational. In large scale computing systems the density of computational devices is in order of tens of thousands of servers. Moreover, the effects of slightest miscalculations and errors are substantial.

The methods and practices that are generally used for design validations are: (a) simulation and (b) testing. The said techniques are useful for small scale networks. However, as the complexity of the systems grow the effectiveness of the aforementioned techniques decreases. Moreover, an alarmingly increasing amount of time is required to uncover the subtle bugs by using testing and simulations. In testing, the program is executed with a set of inputs to evaluate the differences between given input and expected output. The goal in testing is to reduce the frequency of failures. Testing is used to identify the presence of bugs, but it cannot confirm the absence of bugs from the system. In large scale computing systems, such as cloud, the use of testing becomes infeasible as the sizes of such systems are very large. The set of inputs in testing is assumed to cover all possible cases, which involves the range of normal inputs and as well as exceptional scenarios. However, the aforesaid assumptions are not realistic. If we take even a simplest of example of testing a program for adding two real numbers, then there could be infinite number of use cases to test and verify the program correctness. To perform testing and simulation the working prototype of the system or program must exist. First, building a prototype program for LSCS is itself an expensive task. Second, even if the bugs are identified after building a prototype, the cost of fixing bugs at later stages is very high. Therefore, testing and simulations are expensive strategies for the verification of LSCS. The inabilities of traditional tools and techniques to effectively substantiate the working of LSCS have raised questions related to the reliability and robustness. In short, the problem we attempt to solve in this thesis is

"How to design and implement the systems, with a desirable degree of confidence in the correctness and reliability, under different circumstances?"

The pricing model implemented over the large scale computing systems, such as cloud is pay-per-usage, which means that the end-users will pay only for the services usage. Therefore, the specified service level agreement based performance must be provided to the end users to keep up the reputation. If the performance requirements are not met, then the users may not use the services, and the reputation may fall down resulting in a loss of customer and money. Few examples to highlight the impact of performance degradation and errors are: (a) Google reported a 20% revenue loss due to a delay of 500msecs in response time, (b) Amazon reported a sales decrease of 1% due to an additional response time of 100msecs, and (c) Knight Capital Group lost 440 million USD in just 45 minutes, when newly installed trading software went haywire. The aforementioned examples indicate the importance and impact of the performance of the cloud services. Moreover, the due consideration that needs to be given to, and the benefit of, performance to the cloud services are also obvious from the said examples. The traditional methods, such as Testing, are expensive and become infeasible as the sizes of the computing systems are large. Because FM presupposes program semantics that is not considered in Testing and Simulations, the FM techniques are considered more powerful. Moreover, through FMs, users can logically analyze the system to prove properties for any possible inputs. In the said perspective, the use of FM for verifying the functionality and reliability of the systems could be beneficial.

1.4. Research Goals and Objectives

The objective of our research is to deploy various FM tools and techniques to formally analyze the behavior and correctness of the strategies, such as routing algorithms and virtualization models that are implemented in large scale computing systems. The goal of our research is: (a) to thoroughly study the strategies, (b) highlight the grey areas that can be further exploit to increase the reliability and performance, (c) and propose a feasible solution. Compared to conventional random networks, the large scale computing systems, specifically Data Centers (DC) exhibits different architectural characteristics, such as predefined complex architectural and topological pattern composed in different layers. The aforementioned characteristics of the underlying network along with the large scale of the servers situate several challenges towards the application of FMs [6].

1.5. References

[1] R. Buyya, S. Y. Chee, and S. Venugopal, “Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities,” 10th IEEE (HPCC ’08), pp.5-13, Sep. 2008.

[2] P. Mell and T. Grance. Definition of cloud computing. Technical Report, NIST, 2009.

[3] A. Hoonlor, B. K. Szymanski, and M. J. Zaki, “Trends in computer science research,” Communications of the ACM, vol. 56, no. 10, pp. 74-83, 2013.

[4] D. Abadi, “Data management in the cloud: Limitations and opportunities,” IEEE Data Engineering, Bulletin, vol. 32, no. 1, 2009, pp.3–12.

[5] J. M. Wing, “A specifier's introduction to formal methods,” *Computer*, vol. 23, no. 9, pp. 8-22, 1990.

[6] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. Fitzgerald, “Formal methods: Practice and experience,” ACM Computing Surveys (CSUR), vol. 41, no. 4, 2009.

2. RELATED WORK

In this chapter we discussed some of the work that is related to the research we have performed during Ph.D.

2.1. Virtual Machine (VM) Based Cloud Management Platforms and High Level Petri-Nets (HLPN)

Virtualization has been studied extensively in the domain of cloud computing. The research is usually focused towards the security or resource provisioning of VM. In [2.1], to make the VM more secure, the authors have proposed to remove the virtualization layer, while retaining the key features enabled by the cloud. In [2.2], the authors proposed HyperSafe, which is a lightweight approach that endows existing hypervisors with a self-protection capability to provide lifetime flow integrity. In [2.3], authors proposed new security architecture in a hypervisor-based virtualization technology to secure the cloud environment. Similarly, [2.4], also discussed security aspects of VMs in the cloud. In [2.5], a generic model is proposed for resource allocation of VMs in multi-tier distributed environment that describes every VM as a multi-dimensional vector. In [2.6], a two level resource manager is proposed that allocate resources to individual containers using local controllers. Several studies are also available, such as [2.7 - 2.10] that discuss and compare the cloud management platforms. The focus of the said studies was on the discussion and comparison of architecture and feature set of the systems. However, little amount of work has been done in the area of modeling and analysis of cloud management systems, specifically VM-based systems. The HLPN has been widely used for the modeling of systems from various domains of computer science, such as cloud computing, web service framework, Grid infrastructures, scheduling and load balancing. In [2.11], the authors have used Colored Petri Nets (CPN) to model Open Provenance Model (OPM) for the purpose of Model-based Diagnosis in the Cloud (MBD). Virtualization and modeling (including formal analysis and verification) are very rich research domains, when considered separately. However, a diminutive amount of research is performed when both (modeling, including formal analysis and verification of virtualization) are combined.

2.2. Energy Efficient Data Center (DC)

The paradigm shift has occurred in the DCs, where the cost of IT equipment or hardware is no longer the major portion of the overall cost, instead the cost of power and cooling infrastructure has crept in to be the primary cost driver. Thermal imbalance can cause a hurdle towards achieving an efficient operational DC. The presence of the hotspots creates a risk of redlining servers that can cause them to fail prematurely. The power consumption and thermal properties of the devices are directly proportional to each other. Therefore, in this section we will discuss both power and thermal strategies. Several strategies have been proposed to balance the tradeoff between the power, cooling, and performance. There are multiple ways to control the power consumption and thermal properties of the servers, such as through active management of workload hosted on the servers by using admission control strategies, load balancing, and workload migration. The power consumption of the servers can also be tuned through physical control, such as Dynamic Voltage and Frequency Scaling (DVFS) and on-off state control [2.12]-[2.14]. The DVFS has already been implemented in the operating systems, where the CPU utilization drives DVFS controller to adopt the power consumption with the changing workload. A control-theoretic approach to DVFS is proposed in [2.15], where the authors have modified the classical control system algorithm, Proportional Integral Derivative (PID) controller, to perform the dynamic voltage scaling. In [2.16], the author argues that DVFS is not the only solution for processor power management in DC workloads. They propose the use of Per-Core Power Gating (PCPG) for multi-core processors that allows the ability to cut the power supply to selected core, allowing zero power leakage to the gated cores. A technique to control the workload execution on the processor and the power consumption, given some constrained on the temperature of the chip, is proposed in [2.17]. In [2.18], the authors proposed a method to adjust the speed of multi-core processors to maximize the processing with a given set of thermal constraints. They proposed two methods: (a) primal-dual interior-point and (b) dual decomposition, to achieve the desired level of performance under specified thermal constraints. A model-based system, Zephyr, is proposed in [2.19] that combine conventional server power optimization and fan power optimization to optimize overall energy efficiency. The set of blade servers share the cooling capacity of the set of fans, which are controlled by the Multi-Input Multi-Output (MIMO) controller to optimize the aggregate fan power. All of the aforementioned approaches are thermally oblivious, which means that job scheduling and processing decisions are not aware of the heating effect in DC that may cause thermal imbalance and hotspots. Different authors have proposed different thermal aware strategies. Moore et al. [2.20] proposed a temperature aware workload placement approach in DC. The aforesaid approach is based on thermodynamics formulation, power, and thermal profiles of the servers. However, precise measurement of the profiles for such a large number and types of jobs is complicated. Moreover, the thermal and power models are not accurate for DC, as discussed in [2.21]. In another approach [2.22], modeling a thermal topology of DC is discussed that can lead to more efficient workload placement. However, preserving the safe temperature and migration of the resources are not discussed. A DC environmental control system is proposed in [2.23] that use a distributed sensors network to manipulate CRAC units. The control strategy proposed is concentrated to enforce the thermal constraints of cyber infrastructure, while minimizing the heat dissipated by the CRAC unit. The discussion in [2.23] in concentrated only on the CRAC and did not considered the servers. There are other studies, such as [2.24-2.27] that proposed thermal management strategies at a DC level. In [2.28], the authors have modeled DC as a CPS and proposed a control strategy to optimizes the tradeoff between the quality of computational and energy cost. However, the heat recirculation and its effect on the other neighboring nodes are not discussed.

2.3. Formal Verifications of Routing Protocols

A formal verification of ad-hoc routing protocols using SPIN model checker is performed in [2.29]. The authors of [2.29] used Wireless Adaptive Routing Protocol (WARP) to formally verify the real time aspects of the protocol. In [2.30], the authors studied different implementations of Ad-hoc On-demand Distance Vector (AODV) routing protocol. Moreover, to checks C and C++ implementations directly, the authors used their own model checker. A topology approximation algorithm is proposed in [2.31], to tackle the problem of mobility by modeling AODV using colored petri nets. In the paper [2.32], the authors performed specification and verification of LambdaRAM, which is a wide area distributed cache for high performance computing. The authors in [2.32] used TLV for model checking, which uses SMV as an input language. Xiong *et al.* [2.31] have modeled AODV using colored Petri nets (CPN). Some other work towards the verification of routing protocols can be found in [2.33]-[2.35].

2.4. Thermal-Aware Resource Allocation

The cost of IT equipment or hardware is no longer the major portion of the overall cost involves in DCs. Alternatively, the cost of power and cooling infrastructure has crept in to be the primary cost driver. Uneven thermal signatures and hotspots within a DC can lead to hardware failures and energy wastage by the Air Conditioning (AC) units. Moreover, the presence of the hotspots creates a risk of redlining servers that can cause premature failure. The power consumption and thermal properties of the devices are directly proportional. Therefore, in this section we will discuss both the power and thermal strategies.

The topic of energy efficient data centers is addressed by huge number of research communities. The energy efficiency can be achieved in a data center from many dimensions, such as from physical infrastructure perspective and from computational perspective. The aforesaid dimensions are further explored by many researchers to propose new energy efficient strategies. The energy efficiency techniques can be applied to a DC without much overhead and can be broadly categorize as: (a) Dynamic Voltage/Frequency Scaling (DVFS), (b) hot and cold aisle, (c) Dynamic Power Management (DPM), (d) resource allocation, and (e) virtualization [2.36].

As stated above, the power consumption of the servers can be tuned through hardware interfaces, such as DVFS and on-off state control [2.12-2.14]. An integer linear programming modeling approach is proposed in [2.37] that aim to meet the real-time deadlines, while minimizing the hotspots and spatial temperature differences through task scheduling [2.38]. The preceding technique is designed to react when the thermal threshold is approached, instead of avoiding it at a first place. A proactive solution is presented in [2.39] that distribute the workload between cores in a thermally sensitive manner to avoid the temperature to reach the redline value. To predict the temperature, the authors in [2.39] proposed a band-limited predictor that is based on a band limited property of the temperature frequency spectrum. However, in case of mispredictions, the overheads associated with the aforesaid solution are significantly high, as advocated in [2.40]. A scheduling policy is proposed in [2.41] that allocate memory bound tasks to slower frequency processors based on the intensity of memory and current temperature of the processor. Similarly, there are other approaches, such as in [2.42] and [2.43] that attempts to insert additional cycles into the task scheduling process to reduce the thermal signatures of the systems. However, the aforesaid strategies are considered inefficient under many scenarios, where the slack is unavailable between the deadlines. Moreover, the performance is also degraded when the aforesaid approaches are employed, as discussed in [2.40].

To perform the dynamic voltage scaling a control-theoretic approach to the DVFS is proposed in [2.15], where the authors modified the classical control system algorithm, the Proportional Integral Derivative (PID) controller. In [2.16], the author argues that DVFS is not the only solution for processor power management in data center workloads. The authors in [2.16] propose the use of Per-Core Power Gating (PCPG) for multi-core processors that allows the ability to cut the power supply to the selected core, allowing zero power leakage to the gated cores. All of the aforesaid techniques provide a promising control over power management. However, the said techniques can lead to significant negative impact on power management as switching on and off involves overheads [2.44]. Moreover, the approaches are thermally oblivious, where the job scheduling and processing decisions does not account the heating effect in data centers that may cause thermal imbalance and hotspots.

A thermal aware scheduling approach, named as XInt, is proposed in [2.45] that minimizes the inlet temperatures, and leads to minimal heat recirculation and cooling cost for data center operation. A similar scheduling strategy is also proposed in [2.22] to minimize the heat recirculation. However, the aforesaid strategies did not consider the effect of scheduling on the server cooling cost [2.46]. Another approach is proposed in [2.22] that create a thermal topology of the DC to achieve efficient workload placement. However, no discussion is available related to keep the server thermal signatures under the redline values. There are other studies, such as [2.23-2.27] that proposed thermal management strategies using different approaches at a DC level. However, all of the aforesaid studies have not discussed the thermal effect of job allocation on a server and the raise in the temperature on other related servers as a result of ambient effect. In this paper, we analyze a real workload of a DC, using statistical techniques, to observe the thermal impact of job allocation on the selected server and ambient effect on other servers. Moreover, we used the results and findings from the workload analysis, to propose a scheduling scheme that attempts to maintain thermal uniformity within a DC.

2.5. Data Security over the Cloud

Juels *et al.* [2.47] presented a technique to secure the cloud data that provides a number of services, such as integrity, freshness, and availability. The authors employed a gateway application in the enterprise to manage the integrity and freshness checks for the data. The Iris file system is designed to migrate organizations internal file system to the cloud. Moreover, a Merkle tree is used by gateway, which ensures freshness and integrity of data by inserting file blocks, MAC codes, and file version numbers at different levels of the tree. The gateway application also manages the cryptographic keys for confidentiality requirements. Moreover, Ref. [2.47] proposed an auditing framework that audits the cloud environment for ensuring the freshness of the data, data retrievability, and resilience against disk failures. However, the technique heavily depends on the user's employed scheme for data confidentiality. Moreover, data cannot be protected against service provider wholesale.

In [2.48], the authors presented a cryptographic file system that provides confidentiality and integrity services to the outsourced data. The authors used hash based MAC tree for providing the aforesaid services. Block-wise encryption is used for the construction of a MAC tree. The file system at the client side interacts with the file system of the server and outsources the encrypted blocks. Encrypted file blocks and cryptographic metadata are stored separately. Nevertheless, the presence of cryptographic metadata on the storage side can be a potential threat.

The authors in [2.49] proposed a virtual private crypto-graphic storage service to provide confidentiality and integrity to user data within the cloud. The client application in the proposed method has three modules: (a) data processor, (b) data verifier, and (c) token generator. The client application generates a master key to be used for subsequent operations. The data processor encrypts the file to be uploaded with keys generated from the master key and uploads to the cloud. The data download involves the use of token generator that generates a token for the user to download data. Token also contains identity of files to be downloaded. The data verifier checks for the integrity of the data once the data is downloaded from the cloud. Attribute Based Encryption (ABE) is used for encryption. However, the key in [2.49] resides at client side and may be subject to a single point of failure.

A cloud storage system based on secure erasure code is presented in [2.50]. The system uses threshold key servers for storing a user’s key generated by a system manager. User encrypts the data divided into blocks and stores every block on randomly selected multiple servers. The system also provides the functionality of data forwarding by allowing any of the users to forward the data to any other users without downloading. The authors used proxy re-encryption method for forwarding the encrypted data. A similar scheme is presented by the same authors in [2.51] with the difference that the later does not provide data forwarding. However, aforesaid schemes require heavy implementation level changes on the cloud side.

2.6. References

[2.1] E. Keller, J. Szefer, J. Rexford, and R. B. Lee, “NoHype: virtualized cloud infrastructure without the virtualization,” 37th ACM ISCA, pp. 350–361, June 2010.

[2.2] P. Campegiani, F. L. Presti, “A general model for virtual machines resources allocation in multi-tier distributed systems,” International Conference on Autonomic and Autonomous Systems, pp. 162-167, 2009.

[2.3] F. Zhang, J. Chen, H. Chen, and B. Zang, “CloudVisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization,” ACM SOSP, Cascais, Oct. 2011.

[2.4] Z. Wang and X. Jiang, “HyperSafe: A lightweight approach to provide lifetime hypervisor control-flow integrity,” Symposium on Security and Privacy, pp. 380-395, 2010.

[2.5] F. Sabahi, “Secure Virtualization for Cloud Environment Using Hypervisor-based Technology”, Journal of Machine Learning and Computing, vol. 2, no. 1, pp. 39-45.

[2.6] J. Xu, M. Zhao, J. Fortes, R. Carpenter, M. Yousif, “Autonomic resource management in virtualized data centers using fuzzy logic-based approaches,” Journal of Cluster Computing, vol. 11, pp. 213–227, 2008.

[2.7] D. Cerbelaud, S. Garg, and J. Huylebroeck, “Opening the clouds: qualitative overview of the state-of-the-art open source vm-based cloud management platforms,” 10th ACM/IFIP International Conference on Middleware, pp. 1–8, 2009.

[2.8] P. T. Endo, G. E. Gonçalves, J. Kelner, and D. Sadok, “A Survey on Open-source Cloud Computing Solutions,” 8th Workshop on Cloud and Grid Applications, pp. 3-16, 2010.

[2.9] B. Sotomayor, R. S. Montero, I. M. Llorente, I. Foster, “Virtual Infrastructure Management in Private and Hybrid Clouds,” Internet Computing, vol. 13, no. 5, Oct. 2009.

[2.10] N. Khan, A. Noraziah, E. I. Ismail, and M. M. Deris, “Cloud Computing: Analysis of Various Platforms,” Journal of Entrepreneurship and Innovation, vol. 3, no. 2, pp. 51-59, 2012.

[2.11] Y. Li, and O. Boucelma, “A CPN Provenance Model of Workflow: Towards Diagnosis in the Cloud,” Conference on Advances in Databases and Information Systems, pp. 55–64, 2011.

[2.12] Y. Cho and N. Chang, “Energy-aware clock-frequency assignment in microprocessors and memory devices for dynamic voltage scaling,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 26, no. 6, 2007, pp. 1030–1040.

[2.13] H. Aydin and D. Zhu, “Reliability-aware energy management for periodic real-time tasks,” IEEE Transactions on Computers, vol. 58, no. 10, 2009, pp. 1382–1397.

[2.14] P. Choudhary and D. Marculescu, “Power management of voltage/frequency island-based systems using hardware-based methods,” IEEE Transactions on VLSI Systems, vol. 17, no. 3, 2009.

[2.15] A. Varma, B. Ganesh, M. Sen, S. Choudhury, L. Srinivasan, and B. Jacob, “A control-theoretic approach to dynamic voltage scheduling,” International CCASE, pp. 255–266.

[2.16] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis, “Power management of datacenter workloads using per-core power gating,” Computer Architecture Letters, 2009, vol. 8, no. 2, pp. 48–51.

[2.17] Z. Jian-Hui and Y. Chun-Xin, “Design and simulation of the cpu fan and heat sinks,” IEEE Transactions on Components and Packaging Technologies, vol. 31, no. 4, pp. 890–903.

[2.18] A. Mutapcic, S. Boyd, S. Murali, D. Atienza, G. Micheli, and R. Gupta, “Processor speed control with thermal constraints,” IEEE Transactions on Circuits and Systems, vol. 56, no. 9, pp. 1994–2008.

[2.19] N. Tolia, Z. Wang, P. Ranganathan, C. Bash, M. Marwah, and X. Zhu, “Unified power and cooling management in server enclosures,” in InterPACK, pp. 721–730, 2009.

[2.20] J. Moore, J. Chase, P. Ranganathan, and R. Sharma, “Making scheduling "cool": temperature-aware workload placement in data centers,” In USENIX, pp. 61-75, 2005.

[2.21] Q. Tang, S. Gupta, and G. Varsamopoulos, “Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach,” IEEE Transactions on Parallel and Distributed Systems, vol. 19, no. 11, 2008, pp. 1458–1472.

[2.22] J. Moore, J. Chase, and P. Ranganathan, “Weatherman: Automated, online and predictive thermal mapping and management for data centers,” IEEE ICAC, pp. 155-164, 2006.

[2.23] C. Bash, C. Patel, and R. Sharma, “Dynamic thermal management of air cooled data centers,” Thermal and Thermomechanical Phenomena in Electronics Systems, pp. 445–452, 2006.

[2.24] L. Rao, X. Liu, L. Xie, and W. Liu, “Minimizing electricity cost: Optimization of distributed internet data centers in a multi-electricitymarket environment,” International Conference on Computer Communications (INFOCOM), pp. 1–9, 2010.

[2.25] Q. Tang, S. Gupta, and G. Varsamopoulos, “Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 11, 2008, pp. 1458–1472.

[2.26] M. Anderson, M. Buehner, P. Young, D. Hittle, C. Anderson, J. Tu, and D. Hodgson, “MIMO robust control for HVAC systems,” IEEE Transactions on Control Systems Technology, vol. 16, no. 3, 2008, pp. 475– 483.

[2.27] M. Toulouse, G. Doljac, V. Carey, and C. Bash, “Exploration of a potential-flow-based compact model of air-flow transport in data centers,” American Society Of Mechanical Engineers ASME Conference, pp. 41–50, 2009.

[2.28] L. Parolini, N. Toliaz, B. Sinopoli, and B. H. Krogh, “A Cyber-Physical Systems approach to energy management in data centers,” Conference on Cyber-Physical Systems, 2010.

[2.29] R. de Renesse and A. Aghvami, “Formal verification of ad-hoc routing protocols using SPIN model checker”, *12th IEEE Mediterranean Electro technical Conference*, 2004, pp. 1177–1182.

[2.30] D. Engler and M. Musuvathi, “Static analysis versus software model checking for bug finding”, *Verification, Model Checking, and Abstract Interpretation, 5th International Conference*, Lecture Notes in Computer Science, 2004, pp. 191–210.

[2.31] C. Xiong, T. Murata, and J. Tsai, “Modelling and simulation of routing protocol for mobile ad hoc networks using coloured Petri nets”, *Workshop on Formal Methods Applied to Defence Systems in Formal Methods in Software Engineering and Defence Systems*, 2002.

[2.32] V. Vishwanath, L. Zuck, J. Leigh, “Specification and verification of LambdaRAM – a wide-area distributed cache for high performance computing” *6th IEEE/ACM Conference on Formal Methods and Models for Codesign (MEMOCODE) 2008,* USA, June 2008.

[2.33] S. Chiyangwa, M. Kwiatkowska, “A timing analysis of AODV”, *Formal Methods for Open Object-Based Distributed Systems: 7th IFIP WG 6.1 International Conference (FMOODS),* (2005).

[2.34] D. Obradovic, Formal Analysis of Routing Protocols. PhD Thesis, University of Pennsylvania (2002).

[2.35] S. Das, D. L. Dill, "Counter-example based predicate discovery in predicate abstraction", *Formal Methods in Computer-Aided Design, Springer-Verlag*, (2002).

[2.36] E. Masanet, R. Brown, A. Shehabi, J. Koomey, and B. Nordman, "Estimating the energy use and efficiency potential of U.S. data centers," *Proc IEEE*, vol. 99, no. 8, 2011, pp.1440–1453.

[2.37] E. Kursun and C. Y. Cher, "Temperature variation characterization and thermal management of multicore architectures," *IEEE Micro*, vol. 29, pp.116–126, ISSN 0272-1732.

[2.38] J. X. Yang, "Dynamic thermal management through task scheduling," *IEEE Symposium on Performance, Analysis of Systems and Software*, pp. 191–201, 2008.

[2.39] R. Ayoub and K. Indukuri, "Temperature aware dynamic workload scheduling in multisocket CPU servers," *IEEE Transactions on*

Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 9, pp. 1359 –1372, 2011.

[2.40] A. Lewis and N. F. Tzeng, Thermal-Aware Scheduling in Multicore Systems Using Chaotic Attractor Predictors.

[2.41] A. Merkel and J. Stoess, "Resource-conscious scheduling for energy efficiency on multicore processors," *International European conference on Computer systems,* pp. 153–166. 2010.

[2.42] J. Choi and C. Y. Cher, "Thermal-aware task scheduling at the system software level," *ACM Symposium on Low Power Electronics and Design*, pp. 213–218, 2007.

[2.43] P. Bailis and V. J. Reddi, "Dimentrodon: Processor-level preventive thermal management via idle cycle injection," *In Proc. of the 48th*

Design Automation Conference (DAC 2011), June 2011.

[2.44] M. Annavaram, “A case for guarded power gating for multi-core processors,” *In HPCA*, pp. 291-300, 2011.

[2.45] Q. Tang, S. K. Gupta, and G. Varsamopoulos, “Thermal-aware task scheduling for data centers through minimizing heat recirculation,” *IEEE International Conference on Cluster Computing*, pp. 129-138.

[2.46] R. Ayoub, S. Sharifi, and T. S. Rosing, “Gentlecool: Cooling aware proactive workload scheduling in multi-machine systems,” *In Proceedings of the Conference on Design, Automation and Test in Europe* pp. 295-298, 2010.

[2.47] A. Juels and A. Opera, “New approaches to security and availability for cloud data,” *Communications of the ACM*, Vol. 56, No. 2, 2013, pp. 64-73.

[2.48] A. Yun, C. Shi, and Y. Kim, “On protecting integrity and confidentiality of cryptographic file system for outscored storage,” *Proceedings of 2009 ACM workshop on cloud computing security CCSA’09*, pp. 67-76, 2009.

[2.49] S. Kamara and K. Lauter, “Cryptographic cloud storage,” *Financial Cryptography and Data Security,* Springer Berlin Heidelberg, 2010, pp. 136-149.

[2.50] H. Lin and W. Tzeng, “A secure erasure code-based cloud storage system with secure data forwarding,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 6, June 2012, pp. 995-1003.

[2.51] H. Lin and W. Tzeng, “A secure decentralized erasure code for distributed network storage,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 11, Nov. 2010, pp. 1586-1594.

3. MODELING AND ANALYSIS OF STATE-OF-THE-ART VM-BASED CLOUD MANAGEMENT PLATFORMS

This paper is published in *IEEE Transactions on Cloud Computing (TCC)*, vol. 1, no. 1, pp. 50-63, 2013. The authors of the paper are Saif U. R. Malik, Samee U. Khan, and Sudarshan K. Srinivasan.

3.1. Introduction

The vibrant underlying technology in cloud infrastructure is virtualization that contributes towards the prevalent application and adaptation of cloud computing infrastructure [3.3]. Virtualization can be defined as the process of abstracting the original physical structure of innumerable technologies, such as hardware platform, operating system, a storage device, or other network resources [3.4]. Moreover, machines, applications, desktops, networks, and services are also separated from the underlying physical constraints. The cloud takes virtualization to a step further by using VMs that creates the customer independent system regardless of the underlying hardware in a timely manner. Every physical machine in a cloud can host several VMs, which from a user's perspective is equivalent to a fully functional physical machine. Moreover, VMs can be start and stop anytime without any changes to the underlying hardware. Furthermore, migration of VMs between the physical machines is also possible without much disruption. Therefore, the cloud service providers deploy services on VMs that allow resource provision with more flexibility [3.5, 3.51].

The cloud normally involves a large number of VM and physical machines that makes virtual infrastructure management a cumbersome task. Several solutions are available to cope with the aforementioned problem, such as VMware VirtualCenter, Platform Orchestration, and

Enomalism that provides an automatic monitoring and deployment of VMs in resource pools [3.6]. Numerous cloud providers, such as Amazon EC2 [3.7], Google App Engine [3.8], and Science Clouds [3.9] uses the aforementioned solutions to manage the virtual infrastructure. Most of the existing cloud computing management platforms are either proprietary or contains software that are not programmable for experimentation purposes. In the said perspective, several open source VM-based cloud management platforms have been launched, such as Eucalyptus [3.10], oVirt [3.11], and Enomaly Elastic Compute Platform (ECP) [3.12], so that researchers from every field can participate towards further development of management platforms in the cloud. Recently, OpenStack [3.25] has attained a significant status in the field of cloud computing. A number of companies that includes some big names, such as IBM, Dell, AMD, and Intel, have joined the OpenStack project.

Several open source IaaS providers have emerged as a result of recent development in open source virtualization [3.13]. Two hypervisors: **(a)** Xen [3.14] and **(b)** KVM [3.15] are the most widely used open source hypervisors in the recent IaaS providers [3.34]. In this paper, we have studied and analyzed three open source state-of-the-art VM-based cloud management platforms: **(a)** Eucalyptus, **(b)** Open Nebula, and **(c)** Nimbus. The said systems have different design interests (as advocated in [3.16]) and that is why we have selected these systems for the study. The differences in the designs make each system suitable for an explicit environment. An important aspect that influences the choice of selecting a particular system for a private cloud is the level of customization. Amongst all of the aforementioned three cloud platforms, Open Nebula provides the highest level of customizability that allow users to switch almost every component from the underlying Virtual Machine Monitor (VMM) to the front-end. Both, the end user and the administrator, relishes the available customization. The customization provided by

Open Nebula is suitable in an experimental environment, where one wants to explore every component and crack new results from the computational perspective. Besides Open Nebula, Nimbus also provides a high level of customization. However, the major portion of customization in Nimbus is available to the administrator. Nimbus is more suitable for an environment, where one is less interested in technical details of the systems, but requires a broad level of customization, such as cooperative scientific communities. Eucalyptus mimics the implementation of Amazon EC2 and is an open source implementation of Amazon Web Service (AWS) API. The customization level is Eucalyptus is very low that makes it appropriate for a private company, where one needs a cloud for own use and wants to avoid mistakes from the users. The suitability of any of the cloud management platforms depends on the requirements of the user or organizations.

Numerous studies are available, such as [3.17], [3.18], [3.19] that discuss and compare the aforementioned cloud management platforms but the previous work largely focused on the architecture and feature set of the systems. Another mutual aspect observed amongst the previous studies is the high level of abstraction, while discussing the architectures of the systems. In this study, we made an effort to diminish the level of abstraction through detailed modeling and formal analysis of the platforms being discussed. We have used High-Level Petri Nets (HLPN) and Z language for the modeling and analysis of the systems. HLPN is used to: **(a)** simulate the systems and **(b)** provide mathematical representation, to analyze the behavior and structural properties of the system. The model of the systems will help analyze: **(a)** the interconnection of the components and processes, **(b)** the fine-grain details of the flow of information amongst the processes, and **(c)** how the information is processed. Moreover, we performed the verification of the models in two-fold. First, we performed the automated verification of the models by following Bounded Model Checking technique using Satisfiability Modulo Theories Library (SMT-Lib) and Z3 solver. To verify using SMT, the petri net models are first translated into SMT along with the specified properties. Then, Z3 solver is used to check either the model satisfies the properties or not. Second, to verify the feasibility of the models as the number of VMs scales, we model about hundred instances of VMs for each platform (Eucalyptus, Open Nebula, and Nimbus) and verify the correctness. The results generated reveals that the models are working correctly. To the best of our knowledge no work has been done to model, analyze, and verify the open source cloud management platforms. This research work will provide the basis for the researchers to understand the design and implementation of the state-of-the-art VM-based cloud platforms. As the inception of the cloud is based on distributed computing (grid and cluster) and virtualization, the research is more inclined towards the computing and storage aspect of the cloud and another crucial aspect of cloud, the connectivity (networking), is usually forgotten [3.13]. This paper will focus on the intercommunication and behavior of the components of the systems rather than the computing and performance measurements.

3.2. Preliminaries

This section will discuss some of the tools and technologies used in this work that will help the reader to understand the topic easily.

3.2.1. High-Level Petri Nets (HLPN)

Petri nets are graphical and mathematical modeling tool that is applicable to many systems characterized as being concurrent, asynchronous, distributed, parallel, non-deterministic, or stochastic [3.20]. In this paper we have used a variant of classical Petri Net model, namely, High-Level Petri Nets (HLPN). (Readers are encouraged to see [3.20], [3.21] for an elaborate introduction to Petri Nets.)

Definition 1: HLPN [3.20], a HLPN is a 7-tuple $N = (P, T, F, \varphi, R, L, M_0)$ where:

1. P is a set of finite places.
2. T is a set of finite transitions such that $P \cap T = \emptyset$.
3. F is a flow relation such that $F \subseteq (P \times T) \cup (T \cup P)$.
4. φ is a mapping function that maps P to data types such that $\varphi: P \rightarrow Type$.
5. R define rules that maps T to predicate logic formulas such that $R: T \rightarrow Formula$.
6. L is a label that maps F to labels such that $L: F \rightarrow Label$.
7. M_0 is the initial marking where $M: P \rightarrow Tokens$.

The first three variables (P, T, F) provides information about the structure of the net and the next three variables $(\varphi, \mathrm{R}, \mathrm{L})$ provides the static semantics, which means the information does not change throughout the system.

The use of HLPN is preferred over Low-level Petri Nets (LLPN) because in LLPN: **(a)** no distinction is available between the tokens, no types or just one type, **(b)** for transition enablement there is no selection of specific tokens even using flow capacity, and **(c)** a place may be viewed as a structural variable, such as array that is not possible to depict.

Let α and β be the nodes of the HLPN N iff $\alpha, \beta \in P \cup T$. A node α is an input node of another node β iff there is a directed arc from α to β such that $(\alpha, \beta) \in F$. Node α is an output node of β iff $(\beta, \alpha) \in F$. For any node $\alpha \in P \cup T$, the pre-condition is $\bullet\, \alpha = \{\beta | (\beta, \alpha) \in F\}$ and post-condition is $\bullet\, \alpha = \{\beta | (\alpha, \beta) \in F\}$.

In HLPN, places can have tokens of different types and can also be a cross product of two or more types, such as in Fig. 3.1 the places are mapped to the types: $\varphi(P1) = \text{Bool}$, $\varphi(P2) = \text{ID}$, $\varphi(P3) = \mathbb{P}(\text{Int})$, $\varphi(P1) = \text{Char}$.

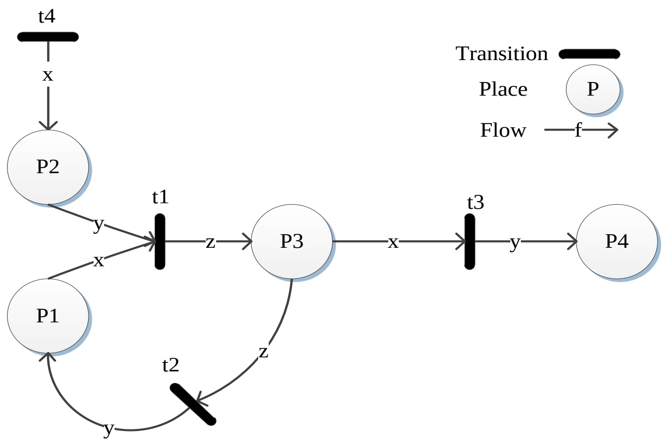


Fig. 3.1. An Example High-Level Petri Net.

The pre-conditions must hold for any transition to be enabled. Moreover, the variables from the incoming flows are used to enable a certain transition. For example, preconditions for *t1* will use *x* and *y* from *P1* and *P2*, respectively. Similarly, post-condition uses variables from outgoing flows for transition firing, and can be written as: $R(t1) = (\mathrm{x} = \mathrm{true}) \wedge (\mathrm{y} \geq 50) \wedge (4 \leq \mathrm{z} \leq 20)$.

3.2.2. SMT-Lib and Z3 Solver

Satisfiability Modulo Theories (SMT) is an area of automated deduction for checking the satisfiability of formulas over some theories of interest and has the roots from Boolean Satisfiability Solvers (SAT) [3.23]. The difference between SMT and SAT is that SMT solvers checks the satisfiability of first-order formulas containing operations from several theories, such as Bit-vector and arithmetic, whereas SAT solvers checks the satisfiability of propositional formulas [3.26]. SMT-Lib provides a common input platform and benchmarking framework that helps in the evaluation of the systems [3.27]. SMT has been used in many fields including deductive software verification. Moreover, recent applications of computer science including planning, model checking, and automated test generation finding, also considers SMT as an important verification tool [3.27]. (Readers are encourage to read [3.28] for the use of SMT-Lib in the verification of OSPF routing protocol [3.50].) Multiple solvers are available that supports SMT-LIB, such as Beaver, Boolector, CVC4, MathSAT5, Z3, and OpenSMT. The solver can be distinguished amongst the features they provide, such as, underlying logic (example first order or temporal), background theories, input formulas, and interface [3.26].

We used Z3 solver in our study, which is a high performance theorem prover developed at Microsoft Research. Z3 is an automated satisfiability checker. Moreover, Z3 also checks whether the set of formulas are satisfiable in the built-in theories of SMT-Lib. Readers are encouraged to see [3.30], for the detailed information about the working and commands of Z3 solver.

3.3. Modeling and Analysis of VM-Based Cloud Management Platforms

VM-based cloud management platforms offer several advantages that include: **(a)** better isolation, **(b)** scalability, **(c)** availability, and **(d)** flexibility. Looking at the benefits provided by the VM-based systems to the cloud a renewed interest of research has emerged in different classes of virtualization, such as desktop, server, application, storage, and network from several industry giants, such as VMware, Red Hat, and Microsoft [3.32]. In this section we will discuss, model, and analyze three VM-based cloud management platforms: **(a)** Eucalyptus, **(b)** Open Nebula, and **(c)** Nimbus.

3.3.1. Components of Open Source Cloud

Before going into the details and modeling of the systems, in this section, we will provide a quick overview of the components of the generic open source cloud computing systems. The components are classified as: **(a)** Hardware and Operating Systems (resides on the physical machine and must be setup properly for any software system to work), **(b)** Network (includes DNS, DHCP, and subnet organization of the physical machines), **(c)** Hypervisor (includes Xen and KVM, which provides framework for VMs to run), **(d)** VM Disk Images (every cloud has a repository of disk images that can be copied and used as a basis for new virtual disks), **(e)** Interface (front-end tools to request VMs and specify parameters), and **(f)** Cloud Framework (includes Eucalyptus or any other framework). The aforementioned components generally make the entire software stack for the cloud computing systems [3.34]. We used the said components in our model to depict the working of the systems.

3.3.2. Eucalyptus

Eucalyptus is an open source VM-based cloud computing management framework that enables users to run and control the instance of virtual machines deployed at several physical resources [3.35]. Eucalyptus was first initiated at the University of California at Santa Barbara and is now supported by the Eucalyptus Systems, Inc [3.36]. The emphasis of the system was to develop an architecture that will allow scientists to experiment cloud related software and architecture. One of the advantages of Eucalyptus is that it uses Amazon Web Service APIs and provides the same interface as of Amazon's EC2 and Simple Storage Service (S3) in a private cluster. Therefore, provides a well-known tool to host and manage VMs.

Installation of Eucalyptus consists of several components: **(a)** Cloud Controller (CLC), **(b)** Cluster Controller (CC), **(c)** Storage Controller (Walrus), and **(d)** Node Controller (NC). The architecture of Eucalyptus (Fig. 3.2) is kept simple, flexible, and hierarchical where every component is implemented as a stand-alone web service. The components implemented as a web services has following benefits [3.35]: **(a)** exposure to a well-defined WSDL document that contains operations being performed and the input/output data structures and **(b)** web features can be extended, such as security policies to secure the communication between components.

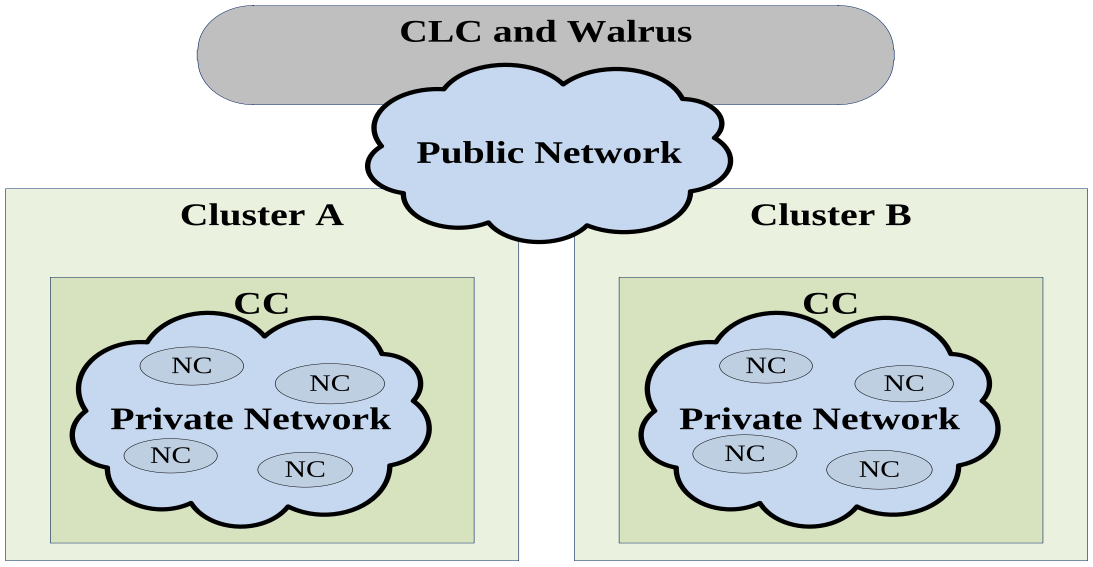


Fig. 3.2. Eucalyptus Architecture.

CLC is the entry point to the cloud that provides configuration interface for managing cluster and instances, Walrus configuration, and user registration. Main responsibilities of CLC includes: **(a)** translation of user initiated commands to CC, **(b)** making high-level scheduling decisions, and **(c)** management of underlying virtualized resources. CC chose the compute node to provision the VM on receiving the command from the CLC. Moreover, gathering information, scheduling VM execution on certain NC, and virtual instance overlay network management for smooth transmission of requests are the responsibilities of CC. Walrus provides a storage service to store virtual machine images and user data. Execution, termination, and inspection of VM instances are performed by the NC. A query is performed by NC to discover the nodes physical resources, such as no. of cores, size of memory, and state of VM instances.

3.3.2.1. Modeling and Analysis

The model of spawning a VM instance in Eucalyptus configuration is illustrated in Fig. 3.3. As stated in Definition 1, the HLPN is a 7-tuple $N = (P, T, F, \varphi, R, L, M_0)$. To begin modeling the system, we first need to specify P and the associated types. As depicted in Fig. 3.3, there are 10

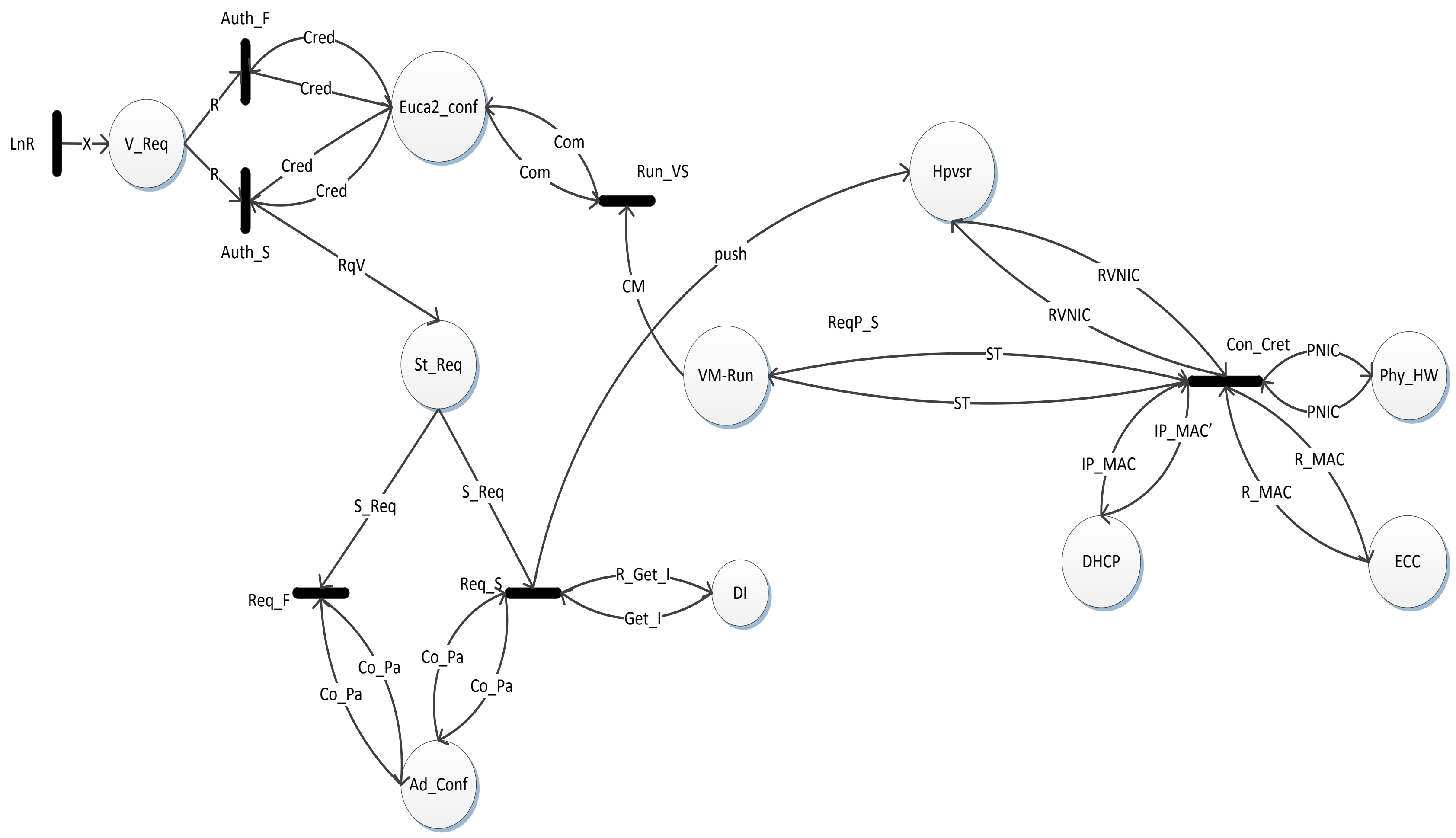


Fig. 3.3. Model of Starting a VM Instance in Eucalyptus.

places in the model. The names and mapping of P are shown in Table 3.1. The types used in the model are illustrated in Table 3.2. The next step is to define the set of rules, pre-conditions, and post-conditions to map to T. Before going to the next step let us have a quick overview of the process of initiating a VM instance. To make a request for an instance of VM the user first needs to configure the front-end. The default front-end is euca2ools, which is similar to the front-end of Amazon EC2. To configure euca2ools the user must download some files along with the keys and instruction. Once, certain environment variables are set the euca2ools is ready to work. The user then uses euca2ools to request the VM. Moreover, the user has to select a configuration for required memory, CPU, and the hard drive space from one of the five preset configurations set by the administrator, for the requested VM. When the head node receives the request, a VM template disk image is extracted from the disk repository and is pushed towards the compute node. The disk image is padded and packaged to be used for the hypervisor. Eucalyptus Cloud Controller (ECC) generates a random MAC address and assigns it to VM instance.

The CC setup a static entry of MAC/IP pair and passes it on to NC. The NC maps virtual NIC of the instance to the physical NIC of the node through network bridging. The instance is initiated on the hypervisor and then the user can directly interact with the VM instance.

We have discussed the process of instantiating the VM and now we can define formulas (pre and post-conditions) to map on transitions. The set of transitions $T = \{LnR, Auth\_F, Auth\_S, Req\_S, Req\_F, Con\_Cret, Run\_VS\}$. New tokens can enter the model only through LnR transition. The rule for the token creation can be stated as: $\boldsymbol{R}(LnR) = \exists t \in T \mid \bullet t = \emptyset$. The next two transitions are $Auth\_S$ and $Auth\_F$, which authenticate the configuration of euca2ools front-end. The said transitions are mapped to the formulas (3.1) and (3.2).

Table 3.1. Places and Mappings of Eucalyptus.

Place	Mapping	Description
φ (V_Req)	*ℙ(Key×Env_Var×CPU×Mem× Disk)*	Holds environment variables and user config.
φ(Euca2_conf)	*ℙ (Key×Env_Var)*	Pre-set config.
φ (St_Req)	*ℙ(CPU×Mem×Disk×Key)*	Intermediate place to hold the configuration values
φ (Ad_Conf)	*ℙ (CPU×Mem×Disk)*	Hold admin config.
φ (DI)	*ℙ (EMI)*	Holds the disk images
φ (Hpvsr)	*ℙ (CPU×Mem×Disk×EMI× NIC×VID ×Key×Env_Var)*	Holds user config. and creates virtual NIC and VM ID
φ (DHCP)	*ℙ (IP × MAC)*	Holds the mappings of IPs to MAC
φ (ECC)	*ℙ (MAC)*	Generate and hold random MAC for VM
φ (Phy_HW)	*ℙ (CPU×Mem×Disk×NIC)*	Holds physical specefication of the system
φ (VM-Run)	*ℙ (CPU×Mem×Disk×EMI×NIC×VID ×IP×MAC×Key×Env_Var)*	Instance of VM is finally created alongwith the specified config.

Table 3.2. Data Types Used in the Model of Eucalyptus.

Types	Description
Key	A string type for euca2ools key authentication
Env_Var	A string type for euca2ools environment variables authentication
CPU	An integer type for the number of CPU/core allocated to the VM.
Mem	A float type for the amount of memory allocated to the VM.
Disk	A float type for the amount of disk space allocated to the VM.
IP	A string type for the IP address of VM
MAC	A string type for the MAC address of the VM
NIC	A string type for NIC of the physical machine
VID	An Integer type for VM ID

$$\boldsymbol{R}\ (Auth\_S) = \forall\, cr \in Cred \mid R[1] = cr[1] \wedge R[2] = cr[2] \wedge \qquad (3.1)$$

$$\forall\, rq \in R \mid RVM := rq \wedge Cred' := Cred$$

$$\boldsymbol{R}\ (Auth\_F) = \forall\, cr \in Cred \mid R[1] \neq cr[1] \wedge R[2] \neq cr[2] \wedge \qquad (3.2)$$

$$Cred' := Cred$$

The formula in (3.1) depicts the success scenario when the euca2ools is able to find both of the credentials (the key and environment variables) in the systems and both are set properly. Similarly, in (3.2) if the euca2ools is unable to locate the specific environment variable or if the key is mismatched, then no further transitions will be fired. After the authentication is successfully performed the next step is to check either the configurations provided by the user for the size of memory, disk, and CPU for the requested VM are same as the ones set by the administrator.

$$\boldsymbol{R}\ (Req\_S) = \forall\, co \in Co\_Pa \mid S\_Req[1] = co[1] \wedge \qquad (3.$$

$$S_{Req[2]} = co[2] \wedge S_{Req[3]} = co[3] \wedge \exists\, rg \in R\_Get\_I, \exists\, cn \in push \mid \qquad 3)$$

$$cn[4] := rg \wedge S\_Req[1] := cn[1] \wedge$$

$$S\_Req[2] := cn[2] \wedge S\_Req[3] := cn[3]$$

$$\boldsymbol{R}(Req\_F) = \forall\, co \in Co\_Pa \mid S\_Req[1] \neq co[1] \wedge \qquad (3.$$

$$S\_Req[2] \neq co[2]\, R[2] \wedge S\_Req[3] \neq co[3] \qquad 4)$$

The administrator configurations reside in the Ad_Conf and the user configurations are placed in St_Req after the transition $Auth\_S$ is fired. In (3.3) and (3.4) both of the configurations (user and administrator) are compared. If (3.3) is fired, then a disk image from a disk image repository is extracted and is transferred to $Hpvsr$ along with the configuration parameters. If (3.4) is fired, then no further transition will be fired because of the configurations mismatch.

Mapping is performed in (3.5), where a random MAC from ECC is generated and assigned to the VM. Moreover, physical NIC from Phy_HW and a virtual NIC from $Hpvsr$ are also mapped. The relation (mapping) between virtual NIC and physical NIC is one-to-many, which means that many virtual $NICs$ can be mapped to one physical NIC. However, the relation between MAC/IP pair and virtual NIC is one-to-one because only a single virtual NIC and a MAC/IP pair can be assigned to a single instance of VM. After all the mapping is completed, the instance of the VM is created with the specified configuration parameters placed at VM_Run. The user can directly interact with the VM instance after it is created using Run_VS.

The design of Eucalyptus supports corporate enterprise computing settings where the administration space is separated from the user space. The users are only allowed to use the system through web interface or specified front end tools. Eucalyptus is easy to deploy on top of the existing resources. Moreover, Eucalyptus is suitable for experimentation because of having modular design and open source in nature.

$$\begin{gathered}\boldsymbol{R}(Con\_Cret) = \{\forall\ im[1]: IP\_MAC \mid \forall\, ma : R\_MAC \mid \forall\, vn[5]: RVNIC, \qquad (3.5)\\ (ma \leftrightarrow im[1]) \leftrightarrow vn[5]\,\} \land \forall\, pn[4] \in PNIC \mid pn[4] \mapsto vn[5]\\ IP\_MAC' = IP\_MAC \cup \{(im[1], ma)\} \land R\_MAC' = R\_MAC \cup \{ma\} \land\\ ST' = ST \cup \{(vn[1], vn[2], vn[3], vn[4],\\ vn[5], vn[6], vn[7], vn[3.8], im[3.1], ma)\}\end{gathered}$$

3.3.3. Open Nebula

Open Nebula was a research project that started in a year 2005 as a management tool for the orchestration and configuration of VMs in datacenter [3.38], [3.39]. Open Nebula is now available as an open source and can be used as a toolkit to build private, public, and hybrid clouds. The key technical aspect of Open Nebula is its architecture that provides a great level of customization and centralization. Moreover, the architecture also supports multiple storage back ends and different hypervisors, such as Xen, VMware, and KVM [3.39]. Shared file system is adopted in Open Nebula for storing all functional and disk images files. The aforementioned exposes the underlying features of libvirt to administrators and users that involves operations, such as VM live migrations. The centralization makes administration of Open Nebula easier. However, one drawback of the default customization with NFS file system is that large amount of space is required to hold all the files.

The architecture of Open Nebula (Fig. 3.4) is divided into three layers: **(a)** Tools, **(b)** Core, and **(c)** Drivers. The first layer contains management tools that can be developed using the Open Nebula core interfaces, such as Command Line Interface, new Open Nebula cloud API, or third party tools that can be created easily using the XML-RPC interface [3.38]. The Open Nebula core performs orchestration and configuration of other components. Moreover, the core also has a set of components that are used to control and monitor VM, virtual networks, hosts,

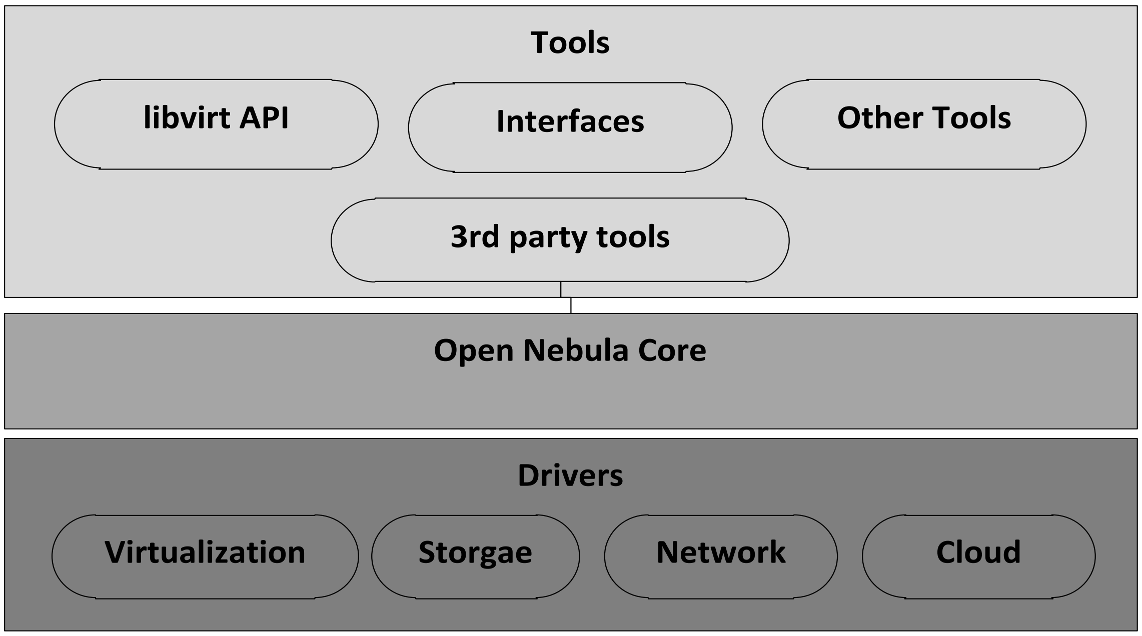


Fig. 3.4. The Architecture of Open Nebula.

and storage. The drivers are pluggable modules that provide a layer of abstraction over the lower level operations, such as virtualization hypervisor, cloud services, and file transfer mechanism. Moreover, the drivers are used by the core layer to perform certain actions, such as cancelling a VM.

3.3.3.1. Modeling and Analysis

The model for initializing the VM using typical Open Nebula configuration is demonstrated in Fig. 3.5. The first step towards modeling the system is to identify the required types, P, and mapping. The types and the descriptions are shown in Table 3.3 and the mapping of P to types is depicted in Table 3.4.

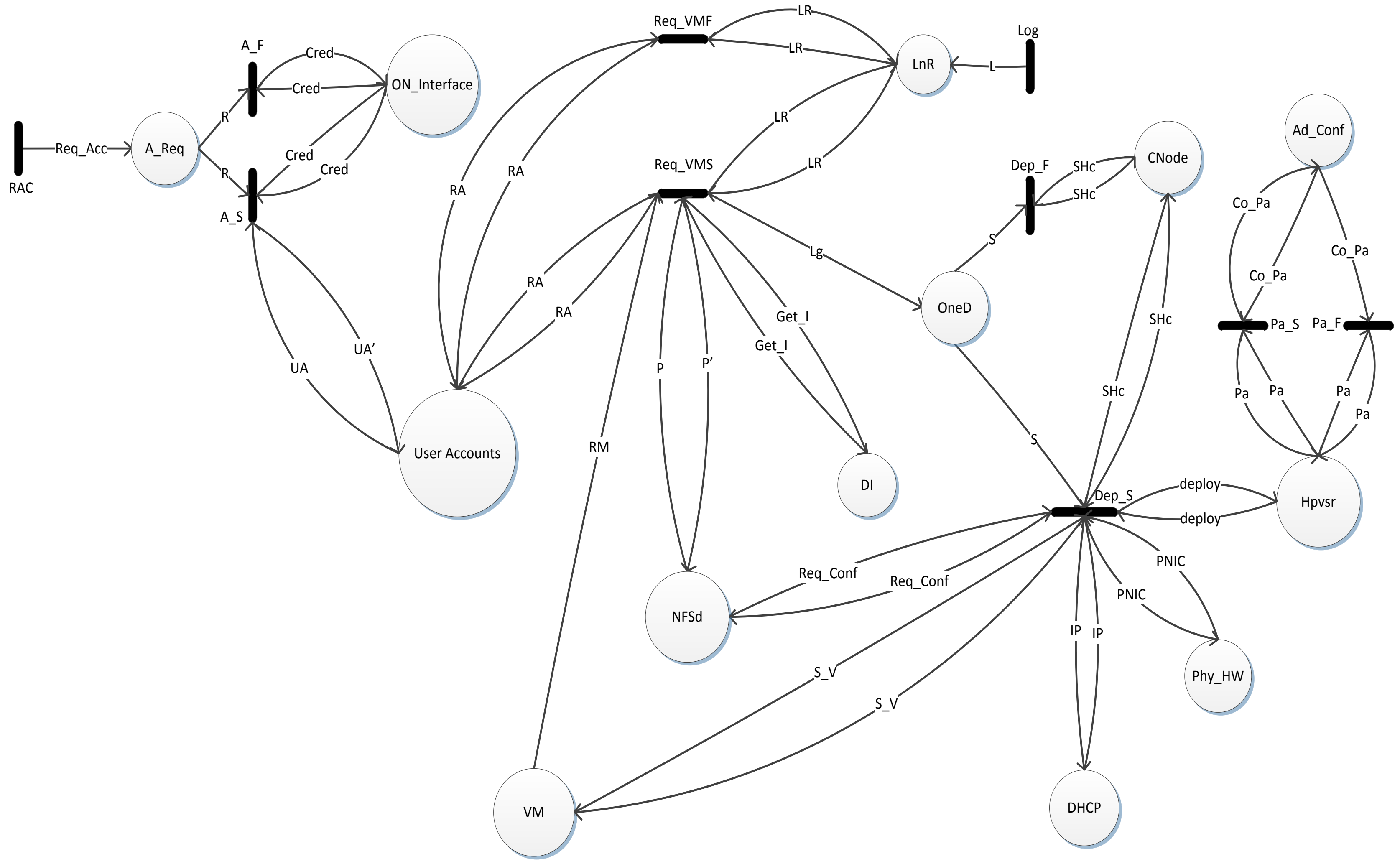


Fig. 3.5. Open Nebula Model for Instantiating a VM.

Table 3.3. Data Types Used in the Model of Open Nebula.

Types	**Description**
Email	A string type for email authentication.
Pass	A string type for password authentication.
UName	A string type for password authentication.
CPU	An integer type for the number of CPU/core allocated to the VM.
Mem	A float type for the amount of memory allocated to the VM.
Disk	A float type for the amount of disk space allocated to the VM.
NMI	Type for the machine image.
SSH_Cert	A string type for the SSH login
SSH_Pass	String type for the SSH encrypted password.
IP	A string type for the IP address of VM
MAC	A string type for MAC address of the VM
NIC	A string type for NIC of the physical machine
VID	An integer type for Virtual Machine ID

To use an Open Nebula cloud the user needs to have an account that Open Nebula provides on demand. After a successful sign up, the user can login with any one of the interface being used, such as sunstone, OCCI, and EC2. The user can request a VM using a command *onevm*, which allow user to manage VMs, such as allocate, deploy, suspend, and shutdown. The NFS directory at the head node holds all the functional and disk image files. As a result of *onevm*, the VM template disk image file is copied from the disk image repository, padded to the required size and configuration, and is saved to the NFS directory.

At that point, the Open Nebula Daemon (oned), which is responsible for the control of VM life-cycle and to coordinate the operations of all modules, logs into the compute node. The compute node provides a virtual NIC, MAC, and mapped it to physical NIC through network bridging. Finally, the instance is created with the specified configurations at the hypervisor. In the previous paragraph we have provided a short overview for the process of instantiating the VM in a typical Open Nebula configuration. Now, we can define formulas (pre and post-conditions) to map on transitions. The set of transitions $T = \{RAC, log, A\_F, A\_S, Req\_VMF, Req\_VMS, Dep\_S, Dep\_F, Pa\_S, Pa\_F\}$.

Table 3.4. Places Used in the Model of Open Nebula.

Places	Mappings	Descriptions
$\varphi\ (A\_Req)$	*ℙ(Email×Pass)*	Holds user requests
$\varphi\ (ON\_Interface)$	*ℙ(Email×Pass)*	Holds existing users
$\varphi\ (LnR)$	*ℙ(Pass×UName× CPU×Mem×Disk)*	Holds user login and config.
$\varphi\ (User\ Accounts)$	*ℙ(Pass×UName)*	Holds user accounts
$\varphi\ (CNode)$	*ℙ(Cert×SSH_Pass)*	Holds login information for oned
$\varphi\ (DI)$	*ℙ(NMI)*	Holds the disk images
$\varphi\ (NFSd)$	*ℙ(CPU×Mem×Disk ×NMI×UName)*	Holds username and user config.
$\varphi\ (OneD)$	*ℙ(Cert×SSH_Pass)*	Holds oned login detail
$\varphi\ (Ad\_Conf)$	*ℙ(CPU×Mem×Disk)*	Holds admin config.
$\varphi\ (Hpvsr)$	*ℙ(CPU×Mem×Disk×NMI× NIC×VID×MAC)*	Hold config., creates virtual NIC, MAC, and VM ID
$\varphi\ (DHCP)$	*ℙ(IP)*	Creates IP for the VM
$\varphi\ (Phy\_HW)$	*ℙ(CPU×Mem×Disk×NIC)*	Holds physical specification of the system
$\varphi\ (VM)$	*ℙ(CPU×Mem×Disk×NMI×NIC×VID×UName×IP)*	VM instance is created alongwith the specified config.

New tokens can only be produced by transition RAC and Log. As seen in Fig. 3.5, no arc is incident on any of the two aforementioned transitions, which is why no pre-condition exists and the rules for the transitions can be written as: $\boldsymbol{R}(RAC) = \exists r \in R \mid \bullet r = \emptyset$ and $\boldsymbol{R}(Log) = \exists lg \in L \mid \bullet lg = \emptyset$. The first step perform by the user is to request an account for an Open Nebula cloud. The transitions A_F and A_S authenticate if the requested user already holds an account or not. The transitions are mapped to the following formulas:

$$\boldsymbol{R}(A\_S) = \forall\, cr \in Cred \mid cr[1] \neq R[1] \wedge \qquad (3.$$

$$UA' := UA \cup \{(R[1], R[2])\} \qquad 6)$$

$$\boldsymbol{R}(A\_F) = \forall\, cr \in Cred \mid \exists\, cr[1] = R[1] \wedge \qquad (3.$$

$$Cred' := Cred \qquad 7)$$

The accounts are created based on the email ID. If the email ID is already associated to an account, then the request is denied. Otherwise, the account is created and the new information is stored in the $User\ Accounts$. The success and failure scenario is depicted in (3.6) and (3.7), respectively. The next step is to login to the Open Nebula cloud and request for the VM.

$$\boldsymbol{R}(Req\_VMS) = \forall\, r \in RA \mid r[1] = LR[1] \wedge r[2] = LR[2] \wedge \qquad (3.8)$$

$$\exists\, np \in P \mid np[1] := LR[3] \wedge np[2] := LR[4] \wedge np[3] := LR[5] \wedge$$

$$np[5] := LR[2] \wedge \forall\, g \in Get\_I \mid np[3.4] := g$$

$$P' = P \cup \{\, (np[1], np[2], np[3], np[4], np[5])\,\}$$

$$\boldsymbol{R}(Req\_VMF) = \forall\, r \in RA \mid r[1] \neq LR[1] \wedge r[2] \neq LR[2] \wedge \qquad (3.9)$$

$$RA' := RA$$

The user account information is stored in $User\ Accounts$. When the user logs in and request for a VM, the login credentials are match and then the command is forwarded, as shown in (3.8) and (3.9). If (3.8) is fired, then the disk image is copied from the disk image repository, padded to correct size and configuration, and is stored in $NFSd$. Moreover, *oned* will login to the compute node only when (3.8) is fired. If (3.9) is fired, then the request for the VM is denied and no further transitions will be fired.

To spawn a VM user provides a configuration file with parameters to be fed into the hypervisor command line. The aforementioned allow users to request for any configuration of memory, disk, and CPU. Therefore, we have performed the authentication of configuration parameters in (3.10) and (3.11) when the hypervisor is generating virtual NIC and MAC. In (3.10), if the configurations provided by the user are same as set by the administrator, then the control is transferred back to $Hpvsr$. Otherwise, (3.11) is fired and the system is terminated.

$$\boldsymbol{R}(Pa\_S) = \forall cp \in Co\_Pa \mid cp[.1] = Pa[1] \wedge \qquad (3.10)$$

$$cp[2] = Pa[2] \wedge cp[3] = Pa[3]$$

$$\boldsymbol{R}(Pa\_F) = \forall cp \in Co\_Pa| \, cp[1] \neq Pa[1] \wedge \qquad (3.11)$$

$$cp[3.2] \neq Pa[2] \wedge cp[3] \neq Pa[3]$$

The $Oned$ process uses Secure Shell (SSH), which is an encrypted network protocol to securely send management functions, to login to the compute node using SSH certificate and password. If (3.12) is fired, then the virtual NIC and MAC from $Hpvsr$ and physical NIC from Phy_HW are mapped using network bridging. The relation between virtual MAC and NIC is one-to-one. The relation between the pair of MAC/NIC and physical NIC is many-to-one, which means one physical NIC can be mapped to many. Once the mapping is completed an IP from $DHCP$ is assigned to the VM and the instance is ready to use. If (3.13) is fired, then the model exits because the SSH certificate or the password provided is incorrect.

$$\boldsymbol{R}(Dep\_S) = \forall sc \in SHc| \, \exists \, sc[1] = S[1] \wedge \qquad (3.12)$$

$$sc[2] = S[2] \wedge \forall rc \in Req\_Conf, \exists d \in deploy|$$

$$d[1] := rc[1] \land d[2] := rc[2] \land d[3] := rc[3] \land d[4] := rc[4] \land$$

$$\{\forall d[5]: deploy \mid \forall d[7]: deploy \mid \forall pn[4]: PNIC,$$

$$(d[7] \leftrightarrow d[5]) \mapsto pn[4]\} \land$$

$$\{\exists\, i : IP \mid s[6] : S\_V, i \leftrightarrow s[6]\}$$

$$deploy' = deploy \cup \{(d[1], d[2], d[3], d[4], d[5], d[6], d[7])\}$$

$$S\_V = S\_V \cup \{(d[1], d[2], d[3], d[4], d[5], d[6], rc[5])\}$$

$$\boldsymbol{R}(Dep\_F) = \forall\, sc \in SHc \mid sc[1] \neq S[1] \land \tag{3.13}$$

$$sc[2] \neq S[2]$$

The level of customization available is Open Nebula is suitable for researchers who wish to combine cloud systems with other technologies. However, to utilize the underlying benefits of the customization the user needs to have some technical expertise. Another downside of the customization is that user can make a mistake while providing configuration for a VM. The centralized nature of Open Nebula makes administration easier. Moreover, higher level of customization makes Open Nebula ideal for research community.

3.3.4. Nimbus

Nimbus is an open source solution that allows clients to lease resources by deploying VM and providing an environment suitable for the user [3.40]. Nimbus is also affiliated with the Globus Project [3.41] and uses Globus credential for user authentication. Nimbus also provides a high level of customization just like Open Nebula. However, the only difference is that much of the customization in Nimbus is restricted to the administrator. Moreover, several components, such as image repository and credentials for user authentication, are kept constant. Furthermore, Nimbus also provides an extensible implementation that supports Web Service Resource Framework (WSRF), Amazon EC2, and other end user services to make a cloud easy to use. A

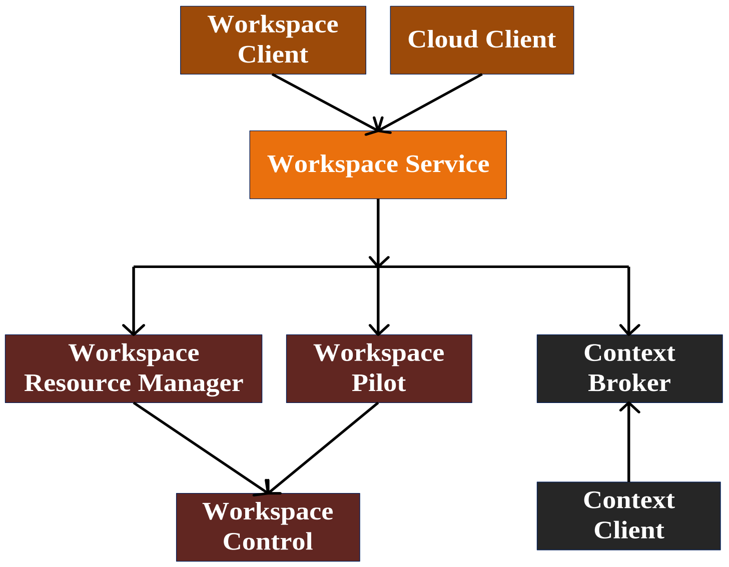


Fig. 3.6. Nimbus Workspace Components.

storage cloud implementation called Cumulus, which is compatible with Amazon Web Service S3 REST API, is included in the Nimbus and is tightly with other central services.

A toolkit is offered to deploy applications on Nimbus that consists of manager service hosting and image repository [3.42]. The components of Nimbus workspace is shown in Fig. 3.6. The *Workspace Service* is a standalone VM manager that can be invoked by remote protocol frontends. Currently, Nimbus supports two frontends: **(a)** EC2 and **(b)** WSRF. The storage service (Cumulus) is also embedded in a Workspace service and can also be installed separately. The *Workspace Resource Manager* deploys and manages workspace nodes. The *Workspace Pilot* allows the integration of preconfigured resources to VMs. Moreover, the aforementioned component also handles signals and has administration tools. The *Workspace Control* is responsible for the management and control of VM instances, disk images, VM integration to a network, and assigning MAC and IP addresses. The *Workspace Client* provides access to the entire feature set of WSRF as a command line client. The aim of *Cloud Client* is to speed up the process of running a VM using instance launches or one-click clusters. The clients can launch large virtual cluster automatically with the use of the *Context Broker*. The Context Client interacts with Context Broker at VM startup and lives on VM.

3.3.4.1. Modeling and Analysis

The model for starting a VM in a typical Nimbus configuration is demonstrated in Fig. 3.7. The first step is to identify the required P and associated types. Table 3.5 depicts the types used in the model and Table 3.6 explains the P and mappings.

The Nimbus uses client known as Cloud Client to interact with the services over multiple protocols. The users first need to download the client and configure it. The use of Cloud Clients makes life easy for the users. The easiest client to use is *Cloud Client* that makes the users up and running in a quick time. The user uses cloud clients to request a VM using explicit credentials. When the head node receives the request, then the VM template disk image is extracted from the repository and pushed to the compute node. The compute node performs the necessary padding and configuration to the image. Virtual MAC and NIC are provided by the compute node using Network bridging. Moreover, the physical NIC and virtual NIC of the VM are also mapped. In Nimbus, every compute node has a DHCP server. The MAC/IP pair is placed in the DHCP server and the VM is ready to be used. The set $T = \{LnR, AcF, AcS, SH\_F, SH\_S, CF, Con, CS\}$.

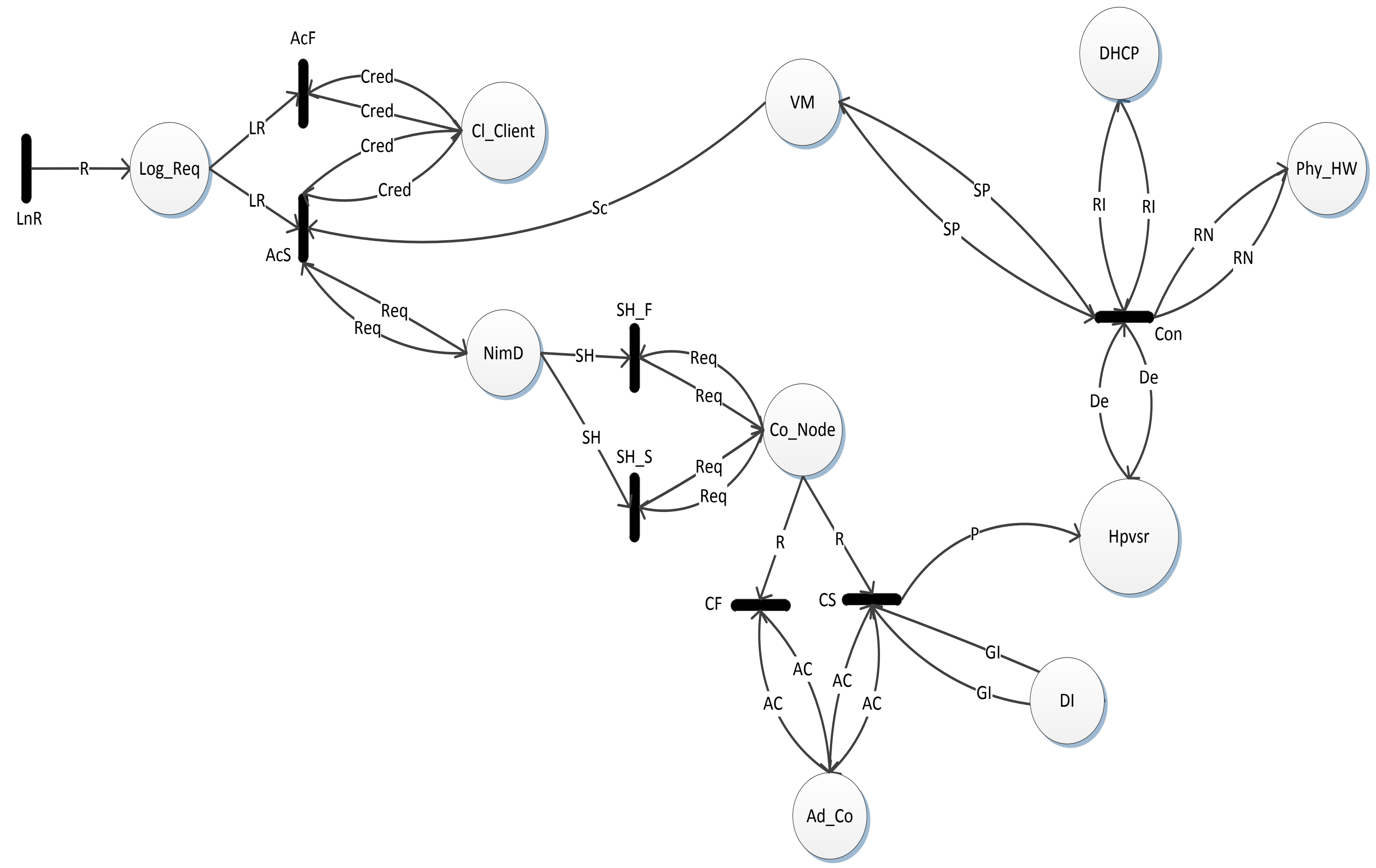


Fig. 3.7. A Model for Nimbus.

Table 3.5. Data Types Used in the Model of Nimbus.

Types	**Description**
UCert	A string type for cloud client authentication.
UKey	A string type for cloud client authentication.
CPU	An integer type for the number of CPU/core allocated to the VM.
Mem	A float type for the amount of memory allocated to the VM.
Disk	A float type for the amount of disk space allocated to the VM.
IP	A string type for the IP address of VM
MAC	A string type for the MAC address of the VM
NIC	A string type for NIC of the virtual/physical machine
VID	An Integer type for Virtual Machine ID
SSH_Cert	A string type for the login authentication of SSH Certificate.
SSH_Pass	A string type for the login authentication of SSH encrypted password.

The seed point of the model from where new tokens are generated is through a transition LnR and the rule for the transition can be stated as: $\boldsymbol{R}(LnR) = \exists t \in T \mid \bullet t = \emptyset$. The transitions AcF and AcS authenticate the credentials of the cloud client. The said transitions are mapped to the following rules:

$$\boldsymbol{R}(AcS) = \forall\, cr \in Cred \mid \exists\, LR[1] \in cr[1] \wedge LR[2] \in cr[2] \wedge \qquad (3.14)$$
$$Cred' := Cred$$

$$\boldsymbol{R}(AcF) = \forall\, cr \in Cred \mid LR[1] \neq cr[1] \wedge \qquad (3.15)$$
$$LR[2] \neq cr[2] \wedge$$
$$Cred' := Cred$$

Table 3.6. Places Used in the Model of Nimbus.

Places	**Mappings**	**Descriptions**
$\varphi\ (Log\_Req)$	*ℙ (UCert×UKey×CPU×Mem× Disk)*	*Holds user credentials and config.*
$\varphi\ (Cl\_Client)$	*ℙ (UCert×UKey)*	*Existing user details*
$\varphi\ (NimD)$	*ℙ(UCert×SSH_Cert×SSH_Pass ×CPU× Mem×Disk)*	*Holds SSH login details and user config*
$\varphi\ (Co\_Node)$	*ℙ(UCert×SSH_Cert×SSH_Pass ×CPU× Mem×Disk)*	*Holds user and SSH login details and specified config.*
$\varphi\ (DI)$	*ℙ (EMI)*	*Holds the disk images*
$\varphi\ (Ad\_Conf)$	*ℙ(CPU×Mem×Disk)*	*Holds admin configurations*
$\varphi\ (Hpvsr)$	*ℙ (UCert×CPU×Mem×Disk× EMI×NIC×MAC×VID)*	*Hold configurations, creates virtual NIC, MAC, and VM ID*
$\varphi\ (DHCP)$	*ℙ (IP×MAC)*	*Hold maps of IPs to MAC*
$\varphi\ (Phy\_HW)$	*ℙ (CPU×Mem×Disk ×NIC)*	*Holds physical specefication of the system*
$\varphi\ (VM)$	*ℙ(UCert×CPU×Mem×Disk× EMI×NIC×MAC×VID×IP)*	*VM instance is created alongwith the specified config.*

The formula in (3.14) represents a success scenario when the cloud client successfully locates both of the credentials (user certificate and user key) and both are configured. In (3.15), if the credentials are mismatched, then no further transitions will be fired and the system will terminate. SSH protocol is used by $NimD$ to login to Co_Node. The SSH_Cert and SSH_Pass are confirmed using rules as stated in (3.16) and (3.17).

$$\boldsymbol{R}(SH\_S) = \forall\, re \in Req \mid \exists\, SH[2] \in re[2] \wedge SH\,[3] \in re[3] \wedge \qquad (3.16)$$

$$Req' := Req \cup \{\, (SH[1], SH[2], SH[3], SH[4], SH[5], SH[6])\,\}$$

$$\boldsymbol{R}(SH\_F) = \forall\, re \in Req \mid SH[2] \neq re[2] \wedge SH\,[3] \neq re[3] \qquad (3.17)$$

$$Req' := Req$$

At that point the configurations for memory, disk, and CPU provided by the user are matched with the administrator configurations. If (3.18) is fired, then the VM template disk image is copied from disk image repository using a distributed storage, such as S3, padded to the correct size, configured, and is pushed to the $Hpvsr$. If (3.19) is fired, then the request will be denied and the system will terminate.

$$\boldsymbol{R}(CS) = \forall\, a \in AC \mid a[4] = R[1] \wedge a[5] = R[2] \wedge a[6] = R[3] \tag{3.18}$$

$$\exists\, g \in GI, \wedge\ pp \in P \wedge$$

$$P' := P \cup \{ (R[1], R[4], R[5] \wedge R[6], g,$$

$$pp[6], pp[7], pp[8]) \}$$

$$\boldsymbol{R}(CF) = \forall\, a \in AC \mid a[4] \neq R[1] \wedge \tag{3.19}$$

$$a[5] \neq R[2] \wedge a[6] \neq R[3] \wedge$$

$$P' := P$$

In the last transition (3.20), an IP from $DHCP$ and virtual MAC and NIC from $Hpvsr$ are mapped one-to-one. Moreover, the physical NIC from Phy_HW is also mapped to virtual NIC and the relation between them is many-to-one, which means that many virtual $NICs$ can be mapped to one physical NIC. The VM is spawned after all the mappings are performed. Nimbus has the ability to provide different resources leases to different users as a mean of scheduling. The flexibility and customization available in Nimbus makes it suitable for scientific community to perform experiment. Moreover, the workspace tools available in Nimbus can operate with Xen hypervisor and as well as with KVM.

$$\boldsymbol{R}(Con) = \{ \forall\ d[6] : De \mid \forall\ d[7] : De \mid \forall\, i[1] : RI \mid \forall\, pn[4] : RN, \tag{3.20}$$

$$(i[1] \leftrightarrow d[7]) \leftrightarrow d[6] \,\} \wedge pn[4] \mapsto d[6]$$

$$RI' = RI \cup \{ (i[7] , d[7]) \}$$

$$De' = De \cup \{ (d[1], d[2], d[3], d[4], d[5], d[6], d[7], d[8]) \}$$

$$RN' = RNSP' = SP \cup \{ (d[1], d[2], d[3], d[4], d[5],$$

$$d[6], d[7], d[8], i[1]) \}$$

3.3.5. OpenStack

The OpenStack is a cloud computing project, launced in July 2010, to provide IaaS. The OpenStack is mainly a collection of three open source projects: **(a)** OpenSTack Compute (Nova), to provide and manege large network of VMs, **(b)** OpenStack Object Storage (Swift), to provide redundant and scalable data storage using cluster of servers, and **(c)** OpenStack Image Service (Glance), to provide discovery, registration, and delivery service for disk images [3.60]. Similar to other systems, such as Eucalyptus and Open Nebula, the OpenStack supports EC2, S3, and Rest Interfaces. The networking between the OpenSTack and Eucalypstus also has certain similarities, such as the automatic bridge creation and IP forwarding for public IPs. Moreover, the authentication process, the development operations (DevOps) deployment tools, and hypervisors, are same between OpenStack and Eucalyptus. Despite the widespread popularity and adaptation of OpenStack, it is still in early stages and will need time to mature, as advocated in [3.29]. The formal analysis, modeling, and verification of the OpenStack are included as a future work in this paper.

3.4. Verification of Models Using SMT-Lib and Z3 Solver

Verification is the process of demonstrating the correctness of an underlying system. Two parameters are required to verify a model or a system: **(a)** specification and **(b)** properties. In this study, we use bounded model checking [3.44] technique to perform the verification, using SMT-

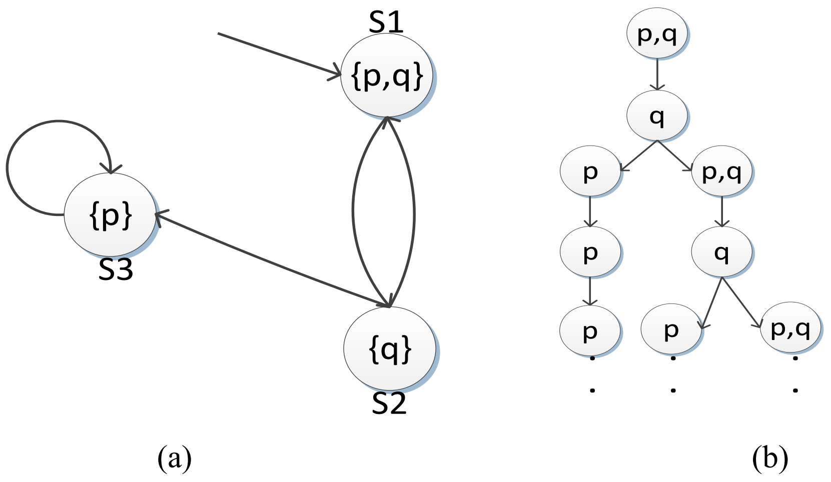


Fig. 3.8. An example of: (a) Kripke Structure and (b) Computational Tree.

Lib and Z3 solver. In bounded model checking, the description of any system is verified, whether any of the acceptable inputs drives the system into a state where the system always terminates after finite number of steps. The process of bounded model checking involves several tasks: **(a)** *Specification*, the description of the system that states the properties or rules, which must be satisfied by the system to be deemed correct, **(b)** *Modeling*, representation of the system, and **(c)** *Verification*, use a tool to check whether the specifications has been satisfied by the model.

Definition 2**:** Bounded Model Checking [3.44], given a Kripke Structure $M = (S, S_0 R, L)$ and a k bound, the bounded model checking problem is to find $\{M \vDash_k Ef\}$ where: S is the finite set of states, S_0 is a set of initial states, R is the set of transitions, such that $R \subseteq S \times S$, L is the set of labels. The bounded model checking problem is to find an execution path in M of length k that satisfies a formula f. Kripke structure, which is a state transition graph, is used to represent the behavior of the system [3.45]. In Kripke structure nodes are the set of reachable states of the system, edge represents the transitions, and label functions map nodes to the set of properties hold in the state. Fig. 3.8 shows an example Kripke structure and computational tree.

Table 3.7. Operators (Op) used in CTL* and Description (Desc).

Op	**Desc**	**Op**	**Desc**
A	For all paths	$\wedge$	Logical AND
E	For some paths	$\vee$	Logical OR
X	Next	$\neg$	Negation
F	For future paths	$\rightarrow$	Implication
G	Globally	$\leftrightarrow$	Double implication

A path in a Kripke structure can be stated as an infinite sequence of states represented as $\rho = S_1, S_2, S_3, \ldots$ such that for $\forall i \geq 0, (S_i, S_{i+1}) \in R$. The model M may produce a path set $=$ $S_1, S_2, S_1, S_2, S_3, S_3, S_3, \ldots$. To describe the property of a model some formal language, such as CTL*, CTL, or LTL is used. As stated in section I, the "*" represents that CTL* is a hybrid LTL and CTL. Some operators used in CTL* are shown in Table 3.7.

To demonstrate the use and meaning of operators an example is provided in Fig. 3.9. The black circle in Fig. 3.10 represents a state p. Moreover, $\mathbf{AF}\boldsymbol{p}$ in Fig. 3.9(a) means, that a future state p is reachable from every path. Furthermore, $\mathbf{AG}\boldsymbol{p}$ in Fig. 3.9(b) means, that state p is globally reachable from every path. (Readers are encouraged to see [3.46] for more details about the CTL*.) For a model to be correct, the states must satisfy the formulas (Definition 2) under a specific bound. The formulas are represented in terms of properties of the systems.

Definition 3: (SMT Solver) [3.31], given a theory $\acute{\Gamma}$ and a formula f, the SMT Solvers perform a check whether f satisfies $\acute{\Gamma}$ or not.

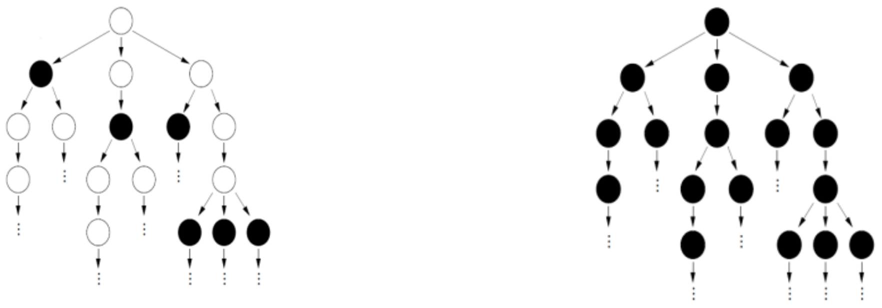


Fig. 3.9. An Example CTL Operators.

To perform the verification of the models using Z3 (an SMT Solver), we unroll the model M and the formula f that provides M_k and f_k, respectively. Moreover, the said parameters are then passed to Z3 to check if $M_k \vDash_\Gamma f_k$ [3.56]. The solver will perform the verification and provide the results as satisfiable (*sat*) or un-satisfiable (*unsat*). If the answer is *sat*, then the solver will generate a counter example, which depicts the violation of the property or formula f. Moreover, if the answer is *unsat*, then formula or the property f holds in M up to the bound k (in our case k is exec. time).

For the models to be correct, the solver should be able to find a terminating state in a model. The failure transitions in all of the models are considered as a terminator of the models. Moreover, the other terminating state in the models is the last state when the instance is successfully created. One property to verify the models is that, *whenever a request is made for a VM and there are no failures, then the instance should be created*. To explain the verification of the models, an example computational tree of a model in Fig. 3.3 is developed and shown in Fig. 3.10. The tree is drawn by following the success transitions only. If we closely analyze, we can see that $VM - Run$ (the final terminating) state is reachable from every path in the tree, which shows that the model terminates after some iterations. Therefore, satisfies the property. In Fig. 3.10, the state $VM - Run$

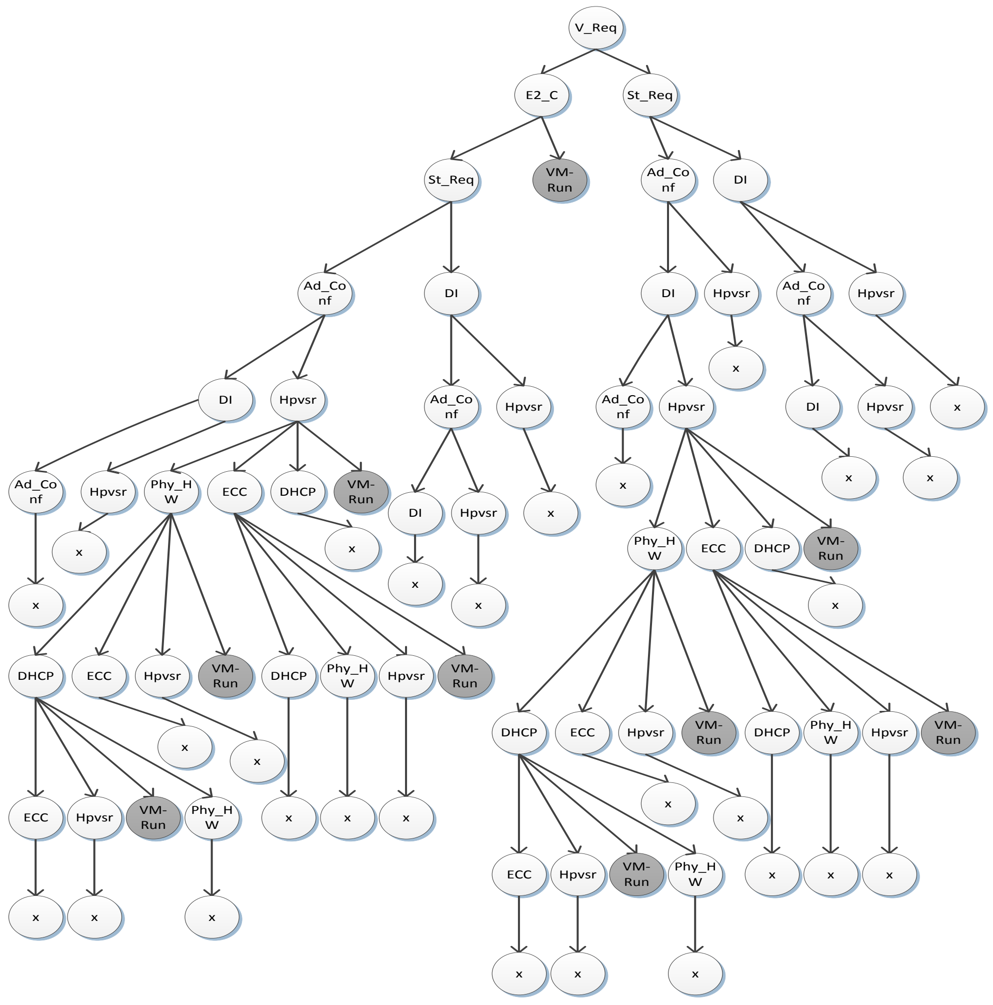


Fig. 3.10. An Example Computational Tree of Eucalyptus.

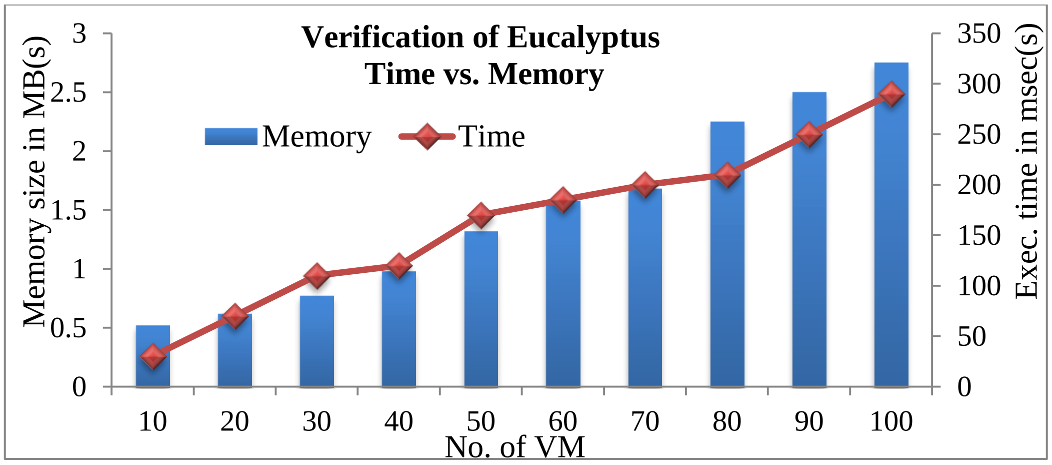


Fig. 3.11. Verification Results of Eucalyptus.

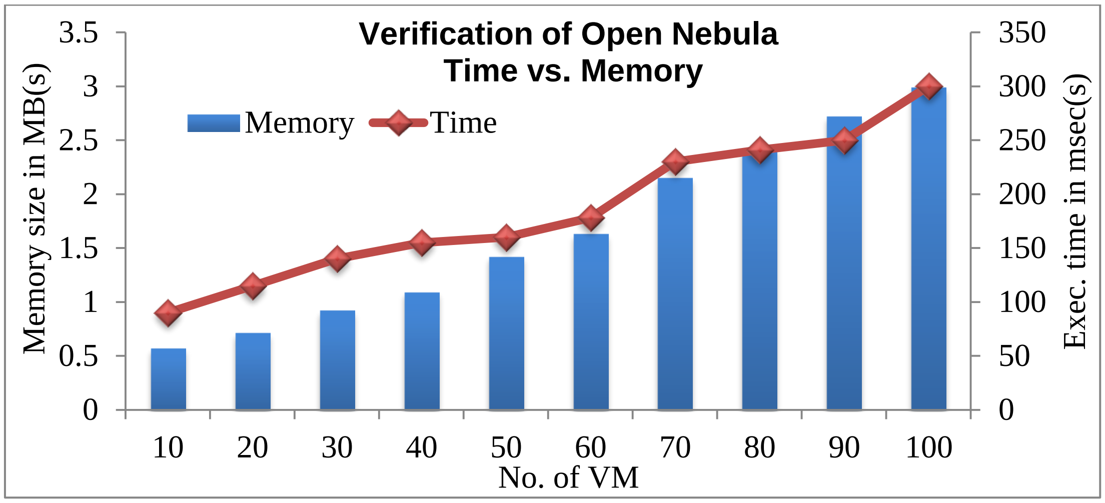


Fig. 3.12. Verification Results of Open Nebula.

at second level shows a scenario when the instance is already been created and user can directly SSH the instance. The states labeled with *"x"* represents, that from the point forward the tree will repeat the predecessor. Similar process has been followed to verify all the models of the systems in this study. We have specified the properties of the VM-based systems in a similar passion and verified whether the properties are satisfied by the models. The properties we have verified for our models are:

1. if a request is made and there are no failures, then the instance of VM must be created,
2. the instance of VM must have the same configurations as requested by the user,

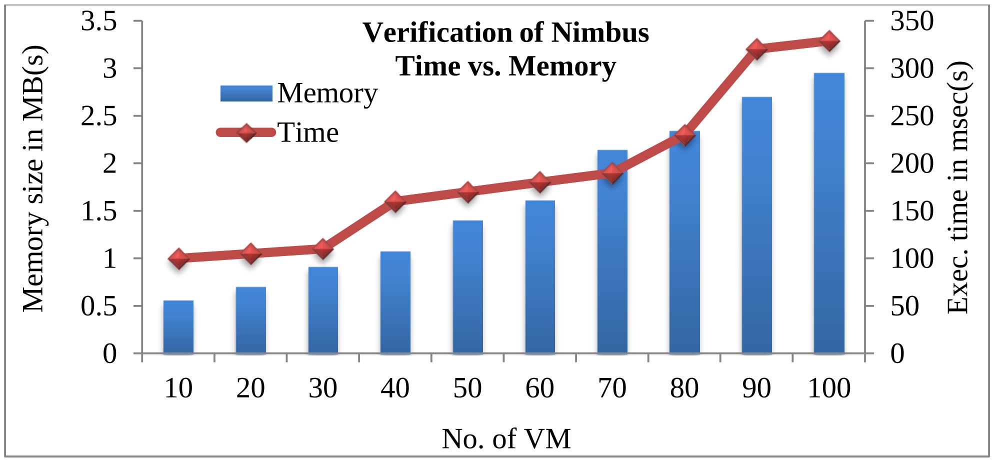


Fig. 3.13. Verification Results of Nimbus.

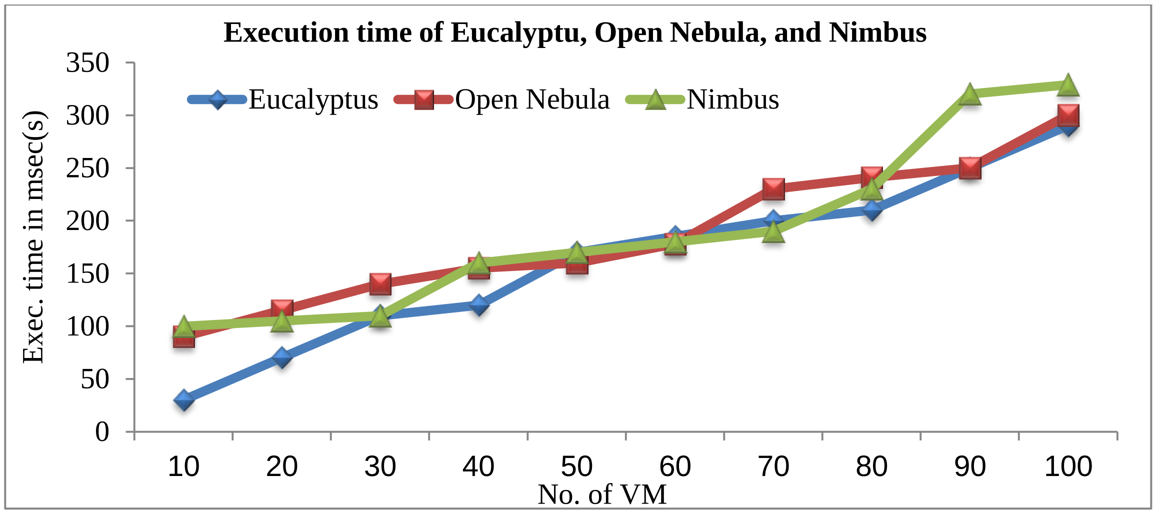


Fig. 3.14. Execution Time Comparison of Eucalyptus, Open Nebula, and Nimbus.

3. the instance should be distinct.

3.4.1. Results

To verify, the models of the VM-based systems are translated to SMT. Moreover, the properties are also translated and specified in SMT. The model along with the properties are given to the Z3 solver to check either the model satisfies the properties or not. If there are no errors, then the model specifications can be stated as correct. Note that our goal in this section is to verify the correctness of the models and not to measure or analyze the performance of the

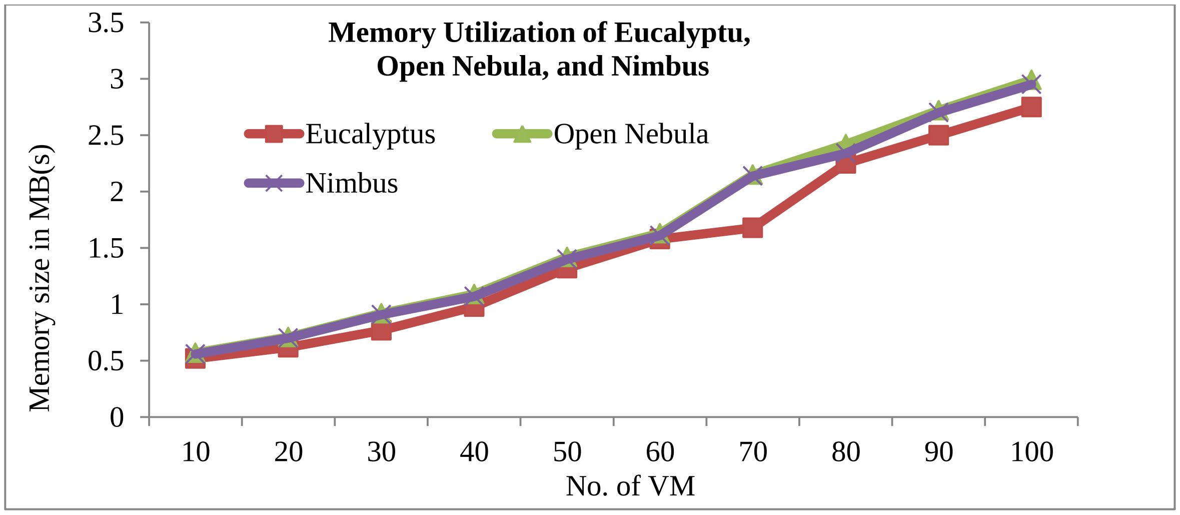


Fig. 3.15. Memory Utilization of the Systems.

systems. Fig. 3.11 depicts the execution time and memory taken to verify the model of Eucalyptus. We have instantiated 100 VMs to verify the properties stated above and to observe the effect of scalability on the working of the models.

The model of Eucalyptus works fine and produces results as expected. The results in Fig. 3.11 shows that an increasing trend is followed in both, the execution time and memory, as the number of VMs increases. The increase in the values is obvious, as the number of VMs will increase more time will be required by the processor to verify and more space will be needed to store the variables.

Fig. 3.12 and 13 depicts the verification results for the model of Open Nebula and Nimbus, respectively. The execution time and memory increase in both of the models as the number of VMs increases. The increase in the values has the same reason that is stated in the results of Eucalyptus. Fig. 3.14 and Fig. 3.15 plot the execution time of all the models (Eucalyptus, Open Nebula, and Nimbus) and memory consumption, respectively. As seen in Fig. 3.15 the memory consumption of Open Nebula and Nimbus has almost similar values, which is due to the fact that Eucalyptus keeps the records of environment variable and configurations. Note that the results indicate the time taken by Z3 solver to verify the models based on the specified properties.

The results in no manner characterize the performance of the models or the systems. Moreover, the goal is to demonstrate the correctness of the models and to highlight the feasibility of the models with respect to scalability and execution time.

3.5. References

[3.1] R. Buyya, S. Y. Chee, and S. Venugopal, “Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities,” 10th IEEE (HPCC ’08), pp.5-13, Sep. 2008.

[3.2] P. Mell and T. Grance. Definition of cloud computing. Technical Report, NIST, 2009.

[3.3] E. Keller, J. Szefer, J. Rexford, and R. B. Lee, “NoHype: virtualized cloud infrastructure without the virtualization,” 37th ACM ISCA, pp. 350–361, June 2010.

[3.4] M. Eisen, Marcum Technology, Introduction to Virtualization, “The Long Island”, Chapter of the IEEE Circuits and Systems (CAS) Society, April 28th, 2011.

[3.5] A. Vichos, Agent-based management of Virtual Machines for Cloud infrastructure, Master’s thesis, School of Informatics, University of Edinburgh, 2011.

[3.6] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, “Capacity Leasing in Cloud Systems using the OpenNebula Engine,” Workshop on Cloud Computing and its Applications 2008 (CCA08), Chicago, USA, Oct. 2008.

[3.7] Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/, accessed 10 Jan. 2013.

[3.8] Google App Engine, https://developers.google.com/appengine/, 10 Jan. 2013.

[3.9] Science Clouds, http://scienceclouds.org/, 10 Jan. 2013.

[3.10] D.Nurmi, R.Wolski, C.Grzegorczyk, G.Obertelli, S. Soman, L.Youseff, and D.Zagorodnov, “The Eucalyptus Open-source Cloud Computing System,” 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2009), Shanghai, China, pp. 124–131, May 2009.

[3.11] oVirt, http://www.ovirt.org/Home, accessed 10 Jan. 2013.

[3.12] Enomaly, http://www.enomaly.com/, accessed 10 Jan. 2013.

[3.13] F. Panzieri, O. Babaoglu, S. Ferretti, V. Ghini, and M. Marzolla, “Distributed Computing in the 21st Century: Some Aspects of Cloud Computing,” in Technology-Enhanced Systems and Tools for Collaborative Learning Scaffolding, Springer, pp. 393-412, 2011.

[3.14] Wojtczuk, R. (2008). Subverting the Xen hypervisor. Black Hat USA, 2008.

[3.15] Habib, I. (2008). Virtualization with kvm. Linux Journal, 2008, vol. 08, no. 166.

[3.16] D. Cerbelaud, S. Garg, and J. Huylebroeck, “Opening the clouds: qualitative overview of the state-of-the-art open source vm-based cloud management platforms,” 10th ACM/IFIP International Conference on Middleware, pp. 1–8, 2009.

[3.17] P. T. Endo, G. E. Gonçalves, J. Kelner, and D. Sadok, “A Survey on Open-source Cloud Computing Solutions,” 8th Workshop on Cloud and Grid Applications, pp. 3-16, 2010.

[3.18] B. Sotomayor, R. S. Montero, I. M. Llorente, I. Foster, “Virtual Infrastructure Management in Private and Hybrid Clouds,” IEEE Internet Computing, vol. 13, no. 5, pp. 14-22, Oct. 2009.

[3.19] N. Khan, A. Noraziah, E. I. Ismail, and M. M. Deris, “Cloud Computing: Analysis of Various Platforms,” Journal of Entrepreneurship and Innovation, vol. 3, no. 2, pp. 51-59, 2012.

[3.20] T. Murata, “Petri Nets: Properties, Analysis and Applications,” Proc. IEEE, vol. 77, no. 4, pp. 541-580, Apr. 1989.

[3.21] Lectures on Petri Nets I: Basic Models, Lecture Notes in Computer Science, vol. 1491, W. Reisig and G. Rozenberg, eds., Berlin: Springer-Verlag, 1998.

[3.22] S. K. Garg, S. Versteeg, and R. Buyya, “A Framework for Ranking of Cloud Computing Services”, FGCS, vol. 29, no. 4, pp. 1012-1023.

[3.23] L. Moura and N. Bjrner, “Satisfiability Modulo Theories: An appetizer,” In Marcel Vinicius Medeiros Oliveira and Jim Woodcock, LNCS, vol. 5902, pp. 23-36, Springer, 2009.

[3.24] B. Javadi, R. Thulasiram and R. Buyya, “Characterizing Spot Price Dynamics in Public Cloud Environments,” FGCS, vol. 29, no. 4, 2013, pp. 988-999.

[3.25] OpenStack, http://www.openstack.org/downloads/openstack-overview-datasheet.pdf, on July, 2013.

[3.26] M. Frade and J. S. Pinto, “Verification conditions for source-level imperative programs,” Technical Report DI-CCTC-08-01, University of Minho, 2008.

[3.27] SMT-Lib http://smtlib.cs.uiowa.edu/, accessed Jan. 2013.

[3.28] S. U. R. Malik, S. K. Srinivasan, S. U. Khan, and L. Wang, “A Methodology for OSPF Routing Protocol Verification,” ScalCom, Changzhou, China, Dec. 2012.

[3.29] G. von Laszewski, J. Diaz, F. Wang, and G. C. Fox, “Comparison of multiple cloud frameworks,” IEEE Conference on Cloud Computing, 2012, pp. 734-741.

[3.30] L. de Moura and N. Bjorner, “Z3: An efficient SMT solver,” International Conference (TACAS ’08), 2008.

[3.31] L. Cordeiro, B. Fischer, and J. Marques-Silva, “SMT-based bounded model checking for embedded ANSI-C software,” ASE, pp. 137–148, 2009.

[3.32] M. Rosenblum, and T. Garfinkel, “Virtual machine monitors: Current technology and future trends,” IEEE Computer, vol. 38, no. 5, pp. 39–47, 2005.

[3.33] Y. Li, and O. Boucelma, “A CPN Provenance Model of Workflow: Towards Diagnosis in the Cloud,” Conference on Advances in Databases and Information Systems, pp. 55–64, 2011.

[3.34] P. Sempolinski and D. Thain, “A comparison and critique of Eucalyptus, OpenNebula and Nimbus,” Cloud-Com, pp. 417-426, 2010.

[3.35] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, “The Eucalyptus open-source cloud-computing system,” IEEE International Symposium on Cluster Computing and the Grid, pp. 124–131, 2009.

[3.36] Eucalyptus Systems home page, http://www.eucalyptus.com, accessed at 15 jan, 2013.

[3.37] J. Xu, M. Zhao, J. Fortes, R. Carpenter, M. Yousif, “Autonomic resource management in virtualized data centers using fuzzy logic-based approaches,” Journal of Cluster Computing, vol. 11, pp. 213–227, 2008.

[3.38] D. Milojicic, I. M. Llorente,; R. S. Montero, “OpenNebula: A Cloud Management Tool,” IEEE Internet Computing, vol. 15, no. 2, pp. 11-14, Mar.-Apr. 2011.

[3.39] Open Nebula, http://www.opennebula.org, 15 Jan, 2013.

[3.40] Nimbus, http://www.nimbusproject.org/docs/2.10, accessed at 16 Jan, 2013.

[3.41] I. Foster, and C. Kesselman, “The Globus Project: A Status Report,” IEEE Heterogeneous Computing Workshop, pp. 4-18, 1998.

[3.42] A Survey of Open-Source Cloud Infrastructure using FutureGrid Testbed, T. Wu, S. N. Baasha, S. S. Karwa, http://salsahpc.indiana.edu/b649proj/proj7.html, accessed at 18 Jan, 2013.

[3.43] P. Campegiani, F. L. Presti, “A general model for virtual machines resources allocation in multi-tier distributed systems,” International Conference on Autonomic and Autonomous Systems, pp. 162-167, 2009.

[3.44] A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu, “Bounded model checking”, Advances in Computers, vol. 58, Academic press, 2003.

[3.45] Y. M. Quemener and T. Jeron, “Model Checking of CL on infinite kripke structures defined by simple grammers,” Technical Report RR-2563, INRIA, France, 1995.

[3.46] M. Maidl, “The common fragment of CTL and LTL,” IEEE Symposium on Foundations of Computer Science, 2000.

[3.47] F. Zhang, J. Chen, H. Chen, and B. Zang, “CloudVisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization,” ACM SOSP, Cascais, Portugal, Oct. 2011.

[3.48] Z. Wang and X. Jiang, “HyperSafe: A lightweight approach to provide lifetime hypervisor control-flow integrity,” IEEE Symposium on Security and Privacy (SP), pp. 380-395, 2010.

[3.49] F. Sabahi, “Secure Virtualization for Cloud Environment Using Hypervisor-based Technology”, International Journal of Machine Learning and Computing, vol. 2, no. 1, pp. 39-45, Feb. 2012.

[3.50] S. U. R. Malik, S. K. Srinivasan, and S. U. Khan, "Convergence Time Analysis of Open Shortest Path First Routing Protocol in Internet Scale Networks," IET Electronics Letters, vol. 48, no. 19, pp. 1188-1190, 2012.

[3.51] J. Kolodiej, S. U. Khan, E. Gelenbe, and E.-G. Talbi, "Scalable Optimization in Grid, Cloud, and Intelligent Network Computing," CCPE, vol. 25, no. 12, pp. 1719-1721, 2013.

4. CONVERGENCE TIME ANALYSIS OF OPEN SHORTEST PATH FIRST ROUTING PROTOCOL IN INTERNET SCALE NETWORKS

This paper is published in *IET Electronics Letters*, vol. 48, no. 19, pp. 1188-1190, 2012. The authors of the paper are Saif U. R. Malik, Sudarshan K. Srinivasan, and Samee U. Khan.

4.1. Introduction

Open Shortest Path First (OSPF) is an adaptive routing protocol to distribute routing information within a single Autonomous System (AS) [4.1]. OSPF divides the network into areas. Each area consists of one or more segments. A segment constitutes the set of routers connected via a common communication channel (example Ethernet). When a failure occurs, topologies are regenerated and paths are recalculated by all of the routers within that area [4.2]. The time a router takes to discover the area topology is known as the convergence time [4.1]. To improve the convergence time of a segment in an area, a router is selected as a Designated Router (DR) on each segment.

Fast convergence time is required to meet network based application demands and Quality of Service (QoS) requirements of modern dynamic large-scale routing domains, such as data centers. Therefore, a lot of effort and studies have been made to improve the performance of OSPF [4.3]. However, the convergence time analysis of OSPF that incorporates DRs has never been studied. We address the aforementioned, by developing a novel method to compute the intra-area convergence time of OSPF-based networks that incorporates DRs, which is the primary contribution of this letter. Moreover, to analyze and benchmark the protocol on Internet scale networks is another contribution of this work. We also show how to use our method to study the effect of: **(a)** DRs, **(b)** cascading failures, and **(c)** topological changes on the convergence time of the routers within an area.

For our experiments, we simulated the detailed implementation of the OSPF protocol based on the specifications reported in [4.2]. To get realistic measurements we generate topologies from BRITE [4.4], using Otter [4.4] (as shown in fig.1.) that represents the exact same characteristics as those of the Internet. The results and analysis provided in the letter will be extremely useful for network administrators seeking to deploy OSPF. Moreover, the results are also useful in the behavioral analysis of OSPF and can provide the basis to reevaluate the design of the protocol to achieve performance optimization.

4.2. Problem Formulation

Consider a network composed of N routers. Let R_i be the i^{th} router, where $1 \leq i \leq N$. A link between two routers R_i and R_j (if it exists) has a communication cost that represents the

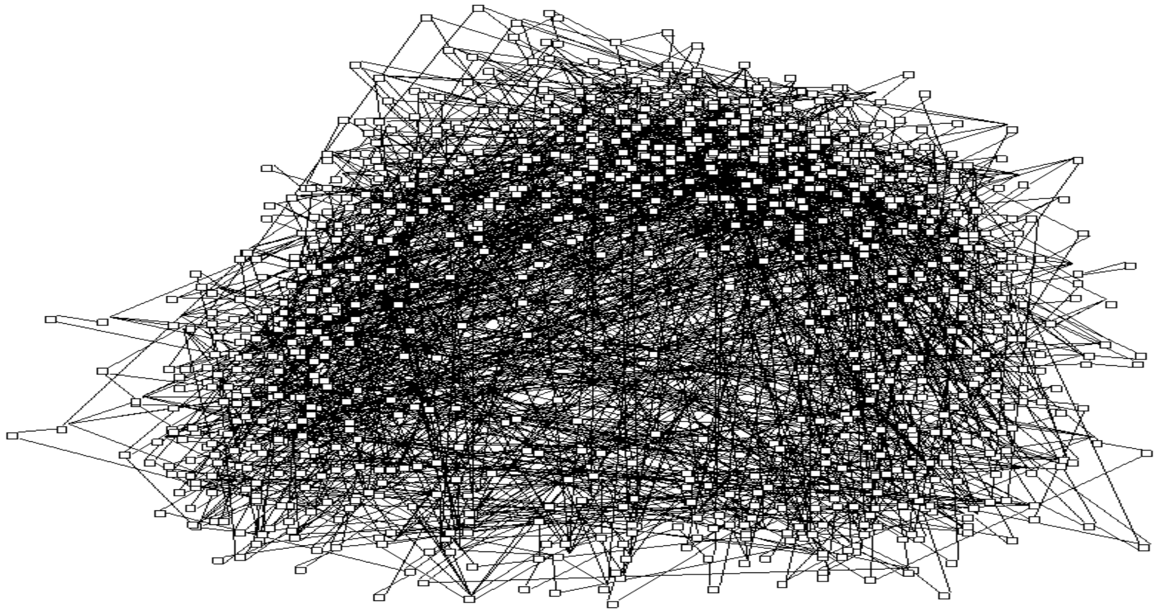


Fig. 4.1. Sample Topology for One Thousand Routers.

minimum time for transferring message from R_i to R_j, which can be represented by the following expression [4.5]:

$$del(R_i, R_j) = \frac{D(R_i, R_j)}{\nu} + \frac{s}{\beta_{ij}}, \tag{4.1}$$

where $D(R_i, R_j)$ is the physical distance between R_i and R_j, v is the propagation delay of the medium (optical fiber in our case), s is the size of the message in kilobytes, and β_{ij} is the available bandwidth between R_i and R_j. If the routers are not directly connected, then the communication cost is the sum of the cost of all links in the shortest path from R_i to R_j. Without the loss of generality, we assume that $del(R_i, R_j) = del(R_j, R_i)$, which is a common assumption in literature [4.5]. Let M be the number of segments within an area and S^k be the k^{th} segment in that area, where $1 \leq k \leq M$. Let DS be the set of DRs within an area and ς^k is the convergence time of S^k. If a failure occurs (could be a link or a router), the routers connected to the failed link or failed router will initiate the updates. $Let\ R_o^k$ be a router that initiates an update in response to a failure. Let Rr be the set of all other routers in the area defined as $Rr = (\cup_{i=1}^{N}\{R_i\}) \backslash \{\{R_o^k\} \cup DS\}$.

R_o^k will detect a failure, if no response is received from a neighboring router for a period longer than the Dead Interval (DI). R_o^k will then update its link state and forward the updated link state to the DR of segment k (represented as d^k). The link state is the description of the interface of the router (IP address of the interface, mask, type of network, routers connected to) and the relationship to other routers. The DR will then flood the information to every other router in the segment after receiving the update. Let R_i^k represent a router that belongs to S^k. The time for R_i^k to receive the update $\left(\psi(R_i^k)\right)$ can be calculated as follows:

$$\psi\left(R_i^k\right) = \begin{cases} DI, & if\ R_i^k = R_o^k \\ \psi_{\forall k \in S: R_i \in k}(d^k) + del\left(d^k, R_i^k\right), & if\ R_i^k \in Rr \end{cases} \quad (4.2)$$

where,

$$\psi(d^k) = \psi_{\forall j \in k: k \in S}\left(R_j^k\right) + del\left(R_j^k, d^k\right). \quad (4.3)$$

We assume that other updates, such as change in bandwidth ($\Delta\beta_{ij}$) are local and incur zero update time. Therefore in (4.2), the value of $\psi\left(R_i^k\right)$ for $R_i^k = R_o^k$, is *DI*. The *DI* of routers is usually four times the "Hello" interval, which is the time between consecutive transmissions of "Hello" packets that are used to indicate the liveliness of nodes. The "Hello" interval is 10 seconds for broadcast and P2P networks, and 30 seconds for all other media [4.2]. The value of $\psi(R_i^k)$ for $R_i^k \in Rr$ is the sum of the time required for d^k to receive updates and the time d^k takes to deliver updates to R_i^k. The value of $\psi(d^k)$ is calculated in (4.3), which is the sum of $\psi\left(R_i^k\right)$ (the node sending the update to d^k) and the communication cost between them, which is given as $del\left(R_j^k, d^k\right)$. Moreover, (4.2) and (4.3) are used to calculate ς^k based on the following equation:

$$\varsigma^k = max\left(\psi(d^k) + del_{\forall j \in k}\left(d^k, R_j^k\right)\right). \quad (4.4)$$

The last router (maximum time taken to receive an update from the corresponding d^k) in S^k that receives the update, determines ς^k. Now, using (4.2), (4.3), and (4.4) the convergence time of an area τ can be calculated as follows:

$$\tau = max_{\forall k \in S}\left((\varsigma^k)\right) + \psi\left((R_o^k) + del(R_o^k, d^k)\right). \quad (4.5)$$

The maximum ς^k amongst all of the segments plus the time when the update is initiated and reaches to the respective DR determines the value of τ.

4.3. Results and Discussions

The value of τ determines the time an area requires to reach a stable (steady) state from an unstable state, which is caused by an update. Therefore, to avoid message losses the network must converge quickly. To this end, we evaluate the effect of: **(a)** the number of DRs, **(b)** cascading failures, and **(c)** topology on the value of τ.

We assume optical fiber as the communication medium having propagation delay $v = 300 \times 10^6 miles/sec$. Ethernet channels have a Maximum Transmit Unit (MTU) of 1500 bytes [4.1]. Also, fragmentation is usually avoided in OSPF [4.1]. Therefore, we assume the message size s to be 1KB (lower than the 1500B cap, but not too low and is typically used in literature for experimentation, such as in [4.6]). The bandwidth value β_{ij} is kept constant at 100Mbps, as advocated in [4.7] for the evaluation purposes. The values of $D(R_i, R_j)$ are assigned from within the range of [1-100] km.

Fig. 4.2 depicts the effect of the number of DRs on the value of τ. To analyze the effect on large and average scale networks, we used N= {1000, 300}. A DR can decrease the segment

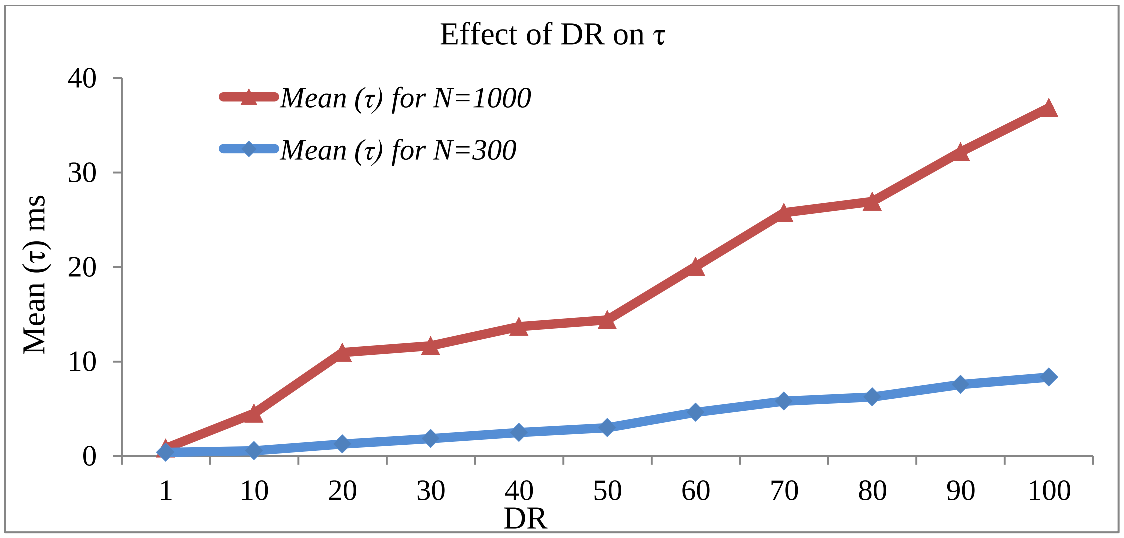


Fig. 4.2. The Effect of DRs on τ.

convergence time from $O(n^2)$ to $O(n)$[4.1]. However, including more DRs in an area has no effect or in some cases may even increase the value of τ. As reported in Fig. 4.2, the mean value of τ increases gradually as the number of DRs increases in the topology. To see why, consider a router *R* under *DR*$_1$. If *DR*$_2$ is added to the area and *R* now falls under *DR*$_2$, then *DR*$_1$ can no longer directly communicate with *R*, but instead is obligatory to communicate via *DR*$_2$. From this example we can see that including a DR can increase the length of communication paths in the area, thereby possibly increasing its convergence time. Therefore, the placement of DR is crucial towards the value of τ.

Fig. 4.3 depicts the effect of cascading failures of routers (also called nodes) and links on τ. The number of DRs in an area is set to one (to avoid the influence of multiple DRs on τ). When a node fails, nearby nodes absorb the load of the failed node. The failed nodes can in turn cause their neighbors to fail (due to overloading) resulting in cascading failures, also known as terminal failure in (communication and power) networks. The degree and placement of a failed node determines its effect on the value of τ. If a failed node or link is in the shortest path of

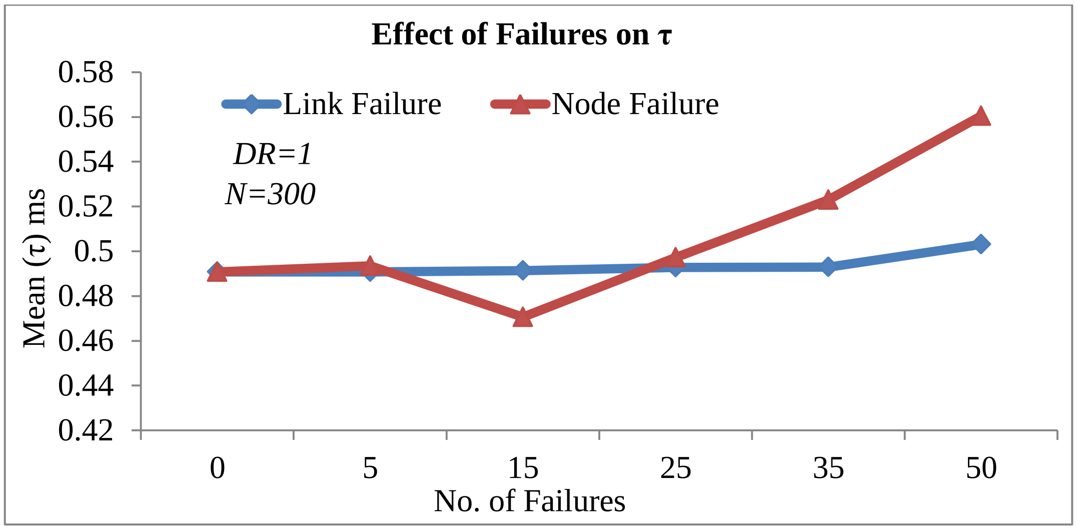


Fig. 4.3. The Effect of Node and Link Failure on τ.

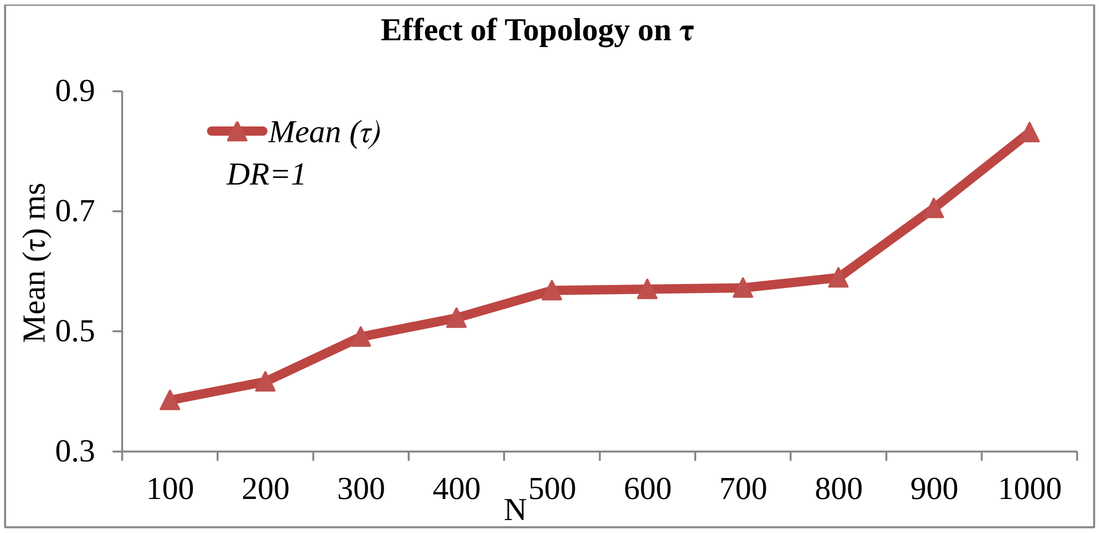


Fig. 4.4. The Effect of Random Topological Changes on τ.

other nodes, then τ may increases. This is because updates to such routers may require a longer path. However, if a failed node or link is: **(a)** a leaf node, a node with low degree, or link on the edge of the topology, or **(b)** not included in the shortest path, then τ can decrease as the failed node need not be updated. Moreover, as can be seen from Fig. 4.3, node failures can affect τ more adversely than link failures. Link failures directly affect only the two routers they are connected to, but node failures affect all its neighbors which is typically more than two.

Fig. 4.4 illustrates the variation of τ due to the changes in the topology, which depicts that the value of τ increases as the number of routers within an area increases. However, an interesting observation is that the value of τ may decrease in certain cases when a router that is included in a topology changes the value of $max(\varsigma^k)$ by adding a new shortest path. To avoid the influence of multiple DRs in an area on τ, the number of DRs is set to one in Fig. 4.4.

4.4. References

[4.1] J. T. Moy, “OSPF; Anatomy of an Internet Routing Protocol,” *Addison-Wesley*, 1998.

[4.2] J. Moy, “OSPF Version 2, The Internet Society OSPFv2,” http://www.ietf.org/rfc/rfc2328.txt, accessed on 08 May, 2012.

[4.3] M, Goyal, M. Soperi, E. Baccelli, G. Choudhury, A. Shaikh, H. Hosseini, “Improving Convergence Speed and Scalability in OSPF: A Survey,” *IEEE Communication Surveys and Tutorials*, 2012, vol. 14, no.2, pp. 443-463.

[4.4] A. Medina, A. Lakhina, I. Matta, and J. Byers, “BRITE: An Approach to Universal Topology Generation,” *MASCOTS*, Ohio, 2001.

[4.5] S. U. Khan and I. Ahmad, “A Pure Nash Equilibrium based Game Theoretical Method for Data Replication across Multiple Servers,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 4, pp. 537-553, 2009.

[4.6] B. Wang, J. Zhang, Y. Guo, and W. Chen, “Fast-Converging Distance Vector Routing Mechanism for IP Networks,” *Journal of Networks*, 2010, vol. 05, no. 9, pp. 1069-1075.

[4.7] R. S. Prasad, M. Murray, C. Dovrolis, and K. Claffy, “Bandwidth Estimation: Metrics, Measurement Techniques, and Tools,” *IEEE Network*, 2003, vol. 17, no. 06, pp. 27-35.

5. MODELING AND ANALYSIS OF THE THERMAL DYNAMICS OF CYBER PHYSICAL DATA CENTERS

This paper is submitted to *IEEE Transactions on Cloud Computing (TCC)* on Sept. 2013. The authors of the paper are Saif U. R. Malik, Kashif Bilal, Samee U. Khan, Bharadwaj Veeravalli, Keqin Li, and Albert Y. Zomaya,

5.1. Introduction

Cloud computing is an emerging paradigm, where a shared pool of resources (networks, servers, storage, applications, and services) can be accessed conveniently, on-demand, and can be rapidly provisioned or released with minimal management effort or service provider interaction [5.1, 5.2]. The Data Center (DC) contributes towards the prevalent application and adoption of cloud by providing architectural and operational foundation. Therefore, the DC serves as a backbone of cloud systems. To maintain a specified Quality of Service (QoS) attributes, such as throughput, the DCs must operate efficiently all the time.

The DC hosts a large number of servers to perform substantial computation and storage. Moreover, to improve the services for high performance computing application, the DC has been increasingly deployed. Blade servers are thin modular servers, usually deployed in DCs that are designed to minimize the use of physical space and energy. Because of the high energy requirements of the computing and cooling devices, the DCs energy consumption can cost millions of dollars. The DC run-time cost is dominated by the cost spent on the energy consumption of computing and cooling technologies [5.3]. According to a report published by Environmental Protection Agency (EPA) [5.4], the peak power consumption of DC in 2006 was 7GW and it was expected to raise 12GW till 2011, leading to a cost of 7.4 billion USD per year.

The income of the DC is defined by the Service Level Agreement (SLA), which defines the amount paid by the users based on the QoS they receive. The computational and operating margins of DCs depend highly on the provision of the QoS. Higher QoS attribute levels lead to higher rates that in turn lead to higher computations. To deliver the specified level of performance, the number of computational devices put in use at all levels of DC has significantly increased. As a result, the rate at which the heat is emitted by the devices has also increased. The cost to stabilize the temperature in the DC has drastically increased and become almost equal to the cost of operating computational systems. The increasing cost of energy consumption calls for new strategies to improve the energy efficiency in DCs. Several strategies have been proposed, such as [5.5, 5.6, 5.7, and 5.8] for efficient energy consumption in the DC. In this paper, we model DC as a Cyber Physical System (CPS) to capture the dynamics and evolution of the thermal properties presented by the DC.

The software aspects, such as scheduling and computations, performed by the devices are modeled as the “Cyber” portion and the supporting infrastructure, such as power supplies and servers, are modeled as the “Physical” portion of the CPS. Several studies are available that model DC as a CPS to achieve energy efficiency, such as [5.9 and 5.10]. The models proposed in the literature are abstract in the sense that they lack detailed analysis of the DC and hence it becomes difficult to exactly understand the dynamics of heat distribution, both from software and infrastructure perspective. Thus, in this paper, we provide a detailed modeling and formulation of the cyber and physical infrastructure, including the heat dissipation of individual components, the heat distribution, and recirculation among the physical portion of the CPS.

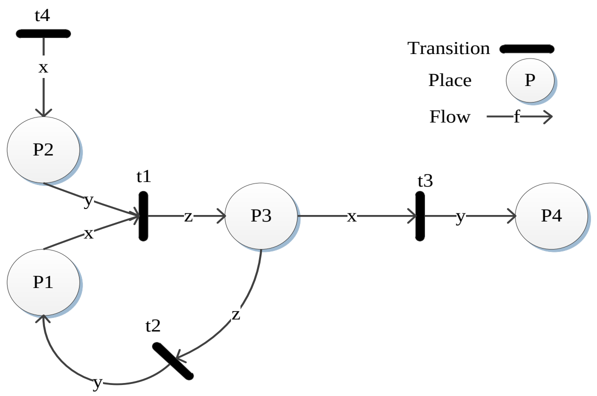


Fig. 5.1. An Example High-Level Petri Net.

The physical infrastructure of the DC follows a hierarchical model, where the computing resources reside at the lowest layer. The network infrastructure can be considered as a multilayer graph, where the servers and switches are vertices and interconnection amongst them are the edges. The servers, access switches, and aggregate switches are assembled in modules (referred as pod) and are arranged in three layers, namely: **(a)** access, **(b)** aggregate, and **(c)** server layer. We perform a thorough analysis and modeling of the thermal subtleties involved at each layer. In doing so, we model heat dissipation of servers, switches (access layer, aggregate layer, and core layer), and the aggregate impact of each component on the overall infrastructure.

By exploiting the thermal dynamics of discrete element, we propose a Thermal Aware Control Strategy (TACS) that uses High Level Centralized Controller (HLCC) and Low Level Centralized Controller (LLCC) to manage and control the thermal dynamics of CPS at different levels, such as: **(a)** low (server) level, **(b)** high (access, aggregate, and core switch) level, **(c)** intra-pod level, and **(d)** inter-pod level. The complete details of all levels and controllers will be discussed in later sections. We perform the simulation of our proposed strategy on a real data center workloads, obtained from Center of Computational Research, State University New York at Buffalo. The traces have more than 22,000 jobs and the records are of one month time. Moreover, we perform a comparative analysis of our proposed strategy with one classical scheduling approach and two thermal aware approaches, namely: **(a)** First Come First Server (FCFS), **(b)** Genetic Algorithm based thermal aware scheduling [5.23], and **(c)** Thermal Aware Scheduling Algorithm (TASA) [5.18].

In this study, we also made an effort to diminish the level of abstraction through detailed modeling and formal analysis of the CPS. We use High-Level Petri Nets (HLPN) and Z language for the modeling and analysis of the systems. The HLPN are used to: **(a)** simulate and **(b)** provide mathematical representation, and (c) analyze the behavior and structural properties of the system. Moreover, we performed the verification of the models using Satisfiability Modulo Theories Library (SMT-Lib) and Z3 solver. We performed the automated verification of the model by following Bounded Model Checking technique using SMT-lib and Z3 solver. To verify using SMT, the Petri net model is first translated into SMT along with the specified properties. Then, Z3 solver is used to check whether the model satisfies the properties or not. The contributions of the paper are as follows:

- modeling DC as a CPS to analyze the thermal dynamics at different levels;
- formulating the thermal properties of major component involved in CPS, the effect of cyber activities on the physical properties of the DC, and vice versa;
- proposing a Thermal Aware Control Strategy (TACS) that uses HLCC and LLCC to manage, control, and coordinate between the cyber and physical portion to maintain unified thermal threshold range;
- conducting simulation and comparison of proposed strategy on a real data center workload and;
- modeling and analyzing the CPS in HLPN, and the verification of the model using SMT-Lib and Z3 Solver.

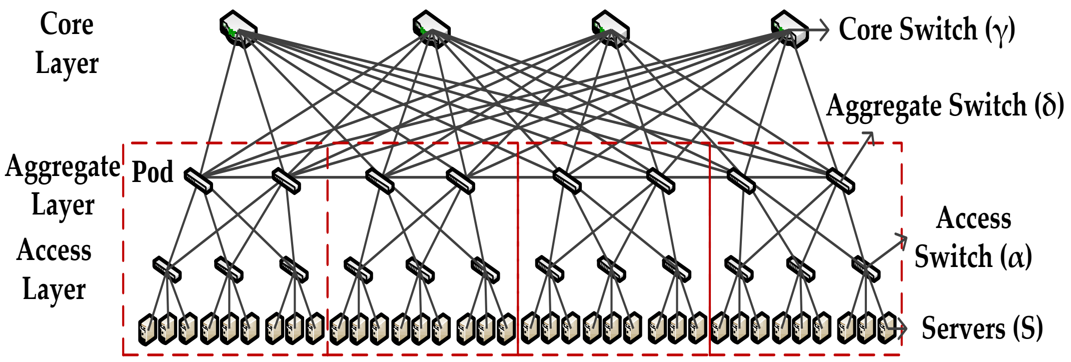


Fig. 5.2. Three-Tier DC Architecture.

5.2. Modeling Thermal Dynamics of Cyber Physical DC

We model DC as a CPS, where the logical classification is made between the computational section and supporting infrastructure. The computational section, such as scheduling, that participates in the distribution, processing, and flow of tasks constitutes the *Cyber* portion. The supporting infrastructure, such as servers, switches, PDUs, and power generators, constitutes the *Physical* portion. The cyber portion performs computations or any other task to deliver the specified QoS attributes. In return, the physical portion emits thermal energy into the DC environment that raises the temperature. In this paper, we present a methodology that analyzes the collective thermal dynamics of cyber and physical portions to maintain a specified range of thermal threshold in the CPS. It is noteworthy, that we are only interested in the thermal dynamics of the DC and not the performance. The DC is logically classified as the combination of the cyber and physical portion:

$$DC = DC(Cyber) + DC(Physical).$$

The CPS is comprised of computing resources, such as servers and the network infrastructure, such as switches, interconnecting all of the computing resources (Fig. 5.2). The CPS follows a hierarchical model, where the computing resources reside at the lowest layer as depicted in Fig. 5.2. The network infrastructure can be considered as a multilayer graph [5.31]. The servers, access switches, and aggregate switches are assembled in modules (referred to as pod) and are arranged in three layers, namely: **(a)** access, **(b)** aggregate, and **(c)** server layer. The core layer is used to connect all of the independent pods together. Note that, the cyber portion resides within the physical portion. Therefore, we model DC in a unified way that can accommodate both, the cyber and physical section. We divided the CPS model into two logical sections: **(a)** Pods (zones) and **(b)** Core Layer Switches, as below:

$$DC = Pod_{\forall i \in k}(i) \cup C_{\forall q \in r}(q), \tag{5.1}$$

where $C(q)$ is the set of core layer switches and r is the total number of core switches (γ) in the network. $Pod(i)$ is the set of pods and k is the total number of pods in the network. Each access layer switch (α) is connected to n number of servers *(S)* in a pod. Moreover, every α is connected to every aggregate switch (δ) in the pod. The number of nodes (including $S, \alpha,$ and δ) in $Pod(i)$ can be calculated as:

$$Pod\ (i) = S^{i}_{(n \times m)} \cup \alpha^{i}_{m} \cup \delta^{i}_{w} \tag{5.2}$$

where $S^{i}_{(n \times m)}$ represents a set of servers connected to α in $Pod\ (i)$. The α^{i}_{m} represents access layer switches in $Pod\ (i)$, where m is the total number of α in $Pod\ (i)$. The δ^{i}_{w} represents aggregate layer switches and w is the number of δ in $Pod\ (i)$. The components in CPS work individually or cooperatively to accomplish the assigned tasks. According to the law of energy conservation, energy can neither be created nor destroyed but it can be converted from one form to another. The mechanical energy is consumed by the physical portion as they perform cyber tasks and almost all the power drawn by the computing devices are dissipated as heat. We model the heat dissipation of every component within the pod, such as $S, \alpha,$ and δ. The heat dissipated by the S is represented as ς_s and can be calculated as follows:

$$\varsigma_s^{i,\alpha} = \left(\varsigma_0 + \varsigma_p + \varsigma_m\right)^{i,\alpha} \tag{5.3}$$

where,

$$\varsigma_p^{i,\alpha} = \left(\varsigma_{rw} + \varsigma_{op}\right)^{i,\alpha} \tag{5.4}$$

The $\varsigma_0^{i,\alpha}$ represents the heat dissipated as a result of the static power to keep the server awake, and $\varsigma_p^{i,\alpha}$ represents the heat dissipation when the processing is being performed. The $\varsigma_0^{i,\alpha}$ is fixed that does not change and is independent. However, $\varsigma_p^{i,\alpha}$ is dynamic and is dependent on the workload. The $\varsigma_m^{i,\alpha}$ represents the heat dissipated by the memory that includes energy consumed during the memory refresh operations. The $\varsigma_p^{i,\alpha}$ is further decomposed into $\varsigma_{rw}^{i,\alpha}$ that represents the heat dissipation because of the read and write operations, and $\varsigma_{op}^{i,\alpha}$ is the heat dissipation as a result of the processing performed. We model switches as normal and high-end switches. The switches used in the core layer are usually high-end switches and dissipate more heat as compared to normal switches. We assume α and δ are normal switches and γ are high-end switches. The heat dissipated by the normal switches, such as α and δ is represented as ξ_n and can be calculated as:

$$\xi_n^{\,i} = \left(\xi_0 + \xi_f + \xi_b + \xi_p\right)^i, \tag{5.5}$$

where,

$$\xi_b = \xi_{ig} + \xi_e, \tag{5.6}$$

and

$$\xi_p = \xi_{p'} + \xi_{pr} + \xi_{rw}. \tag{5.7}$$

The ξ_0 represents the heat dissipation of the switch as a result of static power consumption, ξ_f represents the heat dissipation of the communication fabric used in the switch, ξ_b represents the heat dissipation of the buffer that includes ξ_{ig} and ξ_e, representing the heat dissipation of ingress and egress processing unit, respectively. The ξ_p represents the heat dissipation during the processing that includes $\xi_{p'}$ and ξ_{rw}, representing the static heat dissipation of switch processor, and when read and write operations are performed, respectively. The ξ_{pr} represents the heat dissipation due to the processing performed by the switch. The $\xi_{p'}$ and ξ_0 are constant. However, the ξ_p and ξ_b are dynamic and depend on the workload of the switch.

The γ has different characteristics from α and δ. The α facilitates the connection of the network with the end node devices and for this reason it supports features, such as port security and VLANs. The δ manages or segments the traffic from the leaf nodes into VLANs and provide it to the core layer. For the said reason, δ provides inter-VLANs routing functions to communicate. The γ are the high speed backbone of the network, so they have a very high forwarding rate. Moreover, they have the capability to support link aggregation to ensure adequate bandwidth and traffic routing coming from δ. Furthermore, γ have additional hardware redundancy features, such as redundant power supplies, to swap while the switch continues to operate. Because of the high workload carried out by γ, they dissipate more heat than α and δ. We, represent the heat dissipation of high-end switches (core layer) as $Ж_{\gamma}$, which can be calculated using (5.5), (5.6), and (5.7). However, because of the workload and hardware redundancy the value of $Ж_{\gamma}$ must always be greater than ξ_n. In the previous discussion, we have modeled the heat dissipation of the individual nodes, as in (5.3) and (5.5), involved in the CPS. The heat dissipated by all the servers in $Pod\ (i)$, represented as $§_s^i$, can be calculated as:

$$§_s^i = \sum_{p=1}^{m}\sum_{x=1}^{n}(\varsigma_x^{i,m}) \tag{5.8}$$

where the $\varsigma_x^{i,m}$ represents the heat dissipation of S_x connected to m in $Pod\ (i)$. Moreover, the heat dissipation of all the α and δ in $Pod\ (i)$, represented as $§_\partial^i$ and $§_{g}^i$, respectively, can be calculated as:

$$§_\partial^i = \sum_{x=1}^{m}(\xi_x^i) \tag{5.9}$$

$$§_{g}^i = \sum_{h=1}^{w}(\xi_h^i) \tag{5.10}$$

where ξ_x^i and ξ_h^i represents the heat dissipated by access and aggregate switches in $Pod\ (i)$. Similarly, the overall heat dissipated by the CPS, represented as ψ_c, can be calculated as:

$$\psi_c = \sum_{i=1}^{k}(§_s^i + §_\partial^i + §_{g}^i) + \sum_{j=1}^{r}(Ж_\gamma^j). \tag{5.11}$$

It is noteworthy, that the heat calculations performed at this point, do not consider the ambient effect involved in the CPS environment. The next paragraphs will discuss the dynamics

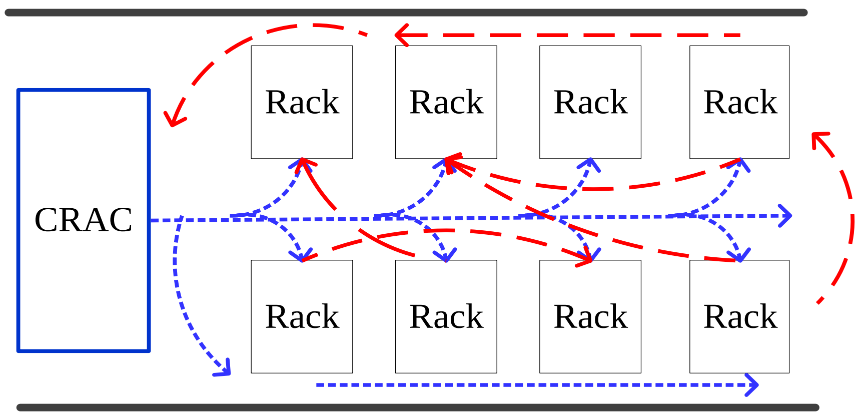


Fig. 5.3. The Ambient Temperature Effect in DC.

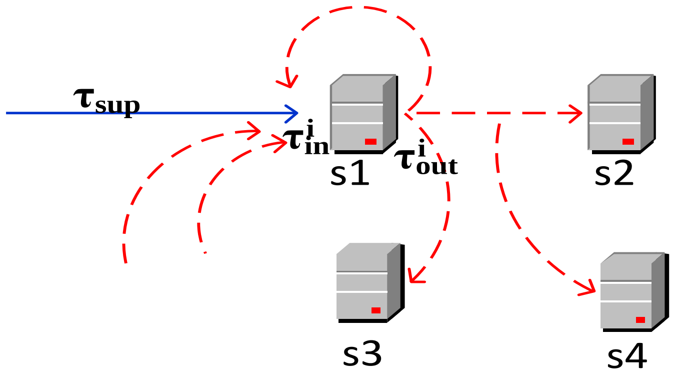


Fig. 5.4. Heat Exchange among Server Nodes.

of ambient temperature and its effect on the heat dissipation of an individual component. The ambient temperature is the surrounding temperature. Figure 3 illustrates the effect of ambient temperature in the CPS environment. The red and blue dotted lines in Fig. 5.3 depict the movement of hot and cold air, respectively. The hot air is exchanged amongst the racks, while the cooling is provided from the cooling devices, such as CRAC. Suppose there are ℵ number of nodes that participate in the heat dissipation of CPS. Two temperatures are associated with each node, the **(a)** input temperature (τ^{i}_{in}) and **(b)** output temperature (τ^{i}_{out}). The τ^{i}_{in} represents the input ambient temperature of node that includes the heat received from other thermal nodes. As depicted in Fig. 5.4, the τ^{i}_{in} of s_1 involves the recirculation (red dotted lines) of hot air from other nodes and cooling temperature (τ_{sup}) from CRAC. The heat dissipated by any node $i \in \aleph$ will change the τ^{i}_{out}. The τ^{i}_{in} and τ^{i}_{out} represent the temperature of the surroundings and not the node. However, the heat dissipated by the node (π^{i}_{out}) can affect the values of τ^{i}_{in} and τ^{i}_{out}. The input temperature of a node (π^{i}_{in}) can be calculated as:

$$\pi^{i}_{in} = \varrho^{i}\left(\tau^{i}_{in}\right) \tag{5.12}$$

where

$$\tau_{in}^{i} = \sum_{j=1}^{\aleph}\left(\pi_{out}^{j}\right) + \tau_{sup}. \tag{5.13}$$

The ϱ is an air coefficient that represents the product of air density (which changes from 1.205*kg/m*^{3} at 20° C to 1.067*kg/m*^{3} at 60° C), heat of air, and flow rate of air. The π_{out}^{i} can be calculated as:

$$\pi_{out}^{i} = \pi_{in}^{i} + \beth^{i} \tag{5.14}$$

where

$$\beth_{i} = \varrho^{i}(\tau_{out}^{i} - \tau_{in}^{i} - \omega^{*}) \tag{5.15}$$

The $\beth_i$ represents the heat dissipation of a node $i \in \aleph$ in proportion to the power consumed during the processing. The ω^* can be replaced by any of the heat dissipation value of three nodes. For instance, if the calculating node is γ, then ω^* can be replaced with Ж. Suppose we have the current power distribution of all the servers in $Pod(i\,)$, represented as a vector $\overrightarrow{P_\iota}$. The temperature profile of all the servers, represented as a vector $\overrightarrow{T_\iota}$, can be calculated based on the given power distribution. The current temperature of S_i in $Pod(j\,)$ is denoted as $t_{cur}^{i,j}$, which can be calculated as, $t_{cur}^{i,j} = \pi_{in}^{i} + \Delta t(c_i)$, where $\Delta t(c_i)$ represents the anticipated change in the temperature cause by executing a task c_i on S. According to the abstract heat model of DC, as discussed in previous works [5.27], the heat distribution and its effect on the surrounding machines can be represented as cross interference coefficient matrix. We follow the same model and compute the heat distribution of the servers using a matrix, represented as $h_{n\times n} = \{\partial_{i,j}\}$, which denotes the thermal effect of S_i on S_j and can be populated as:

$$\partial_{i,j} = \tau_{out}^{i} \times k \times \frac{1}{\hat{h}_j},$$

where k is the thermal conductivity constant of the air and $\hat{h}$ is the hop count of S_j from S_i.

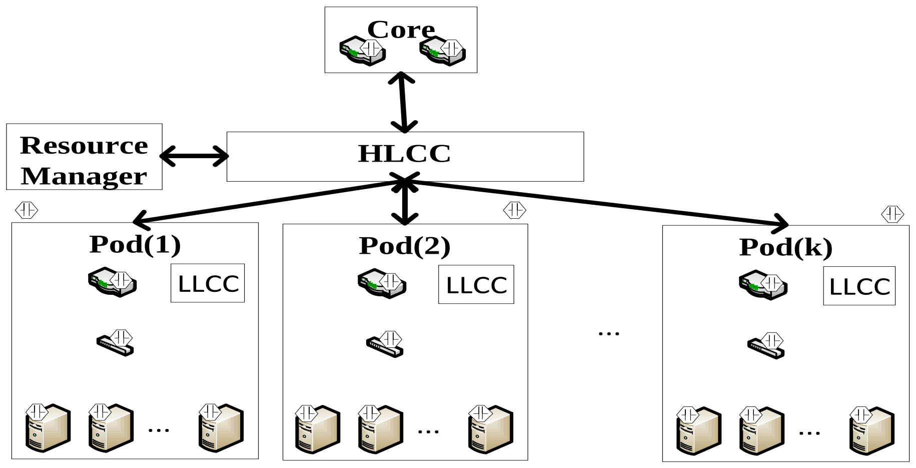


Fig. 5.5. HLCC and LLCC in DC.

5.3. Thermal Aware Control Strategy (TACS)

We propose a thermal aware scheduling approach that uses High Level Centralized Controller (HLCC) and Low Level Centralized Controller (LLCC) to manage and control the thermal dynamics of CPS at different levels, such as: **(a)** low (server) level, **(b)** high (access and aggregate switch) level, **(c)** intra-pod level, and **(d)** inter-pod level. The goal is to eliminate

```
1:  for $i \leftarrow 1$ to $k$ do
2:      $\tau_{\rho}^{i} = \S_{s}^{i} + \S_{\partial}^{i} + \S_{g}^{i}$      // also use in inter-pod migration
3:  end for
4:  Select $min\ (\tau_{\rho}^{i})$
5:  Get $\varsigma_{s}^{i,\alpha}\ \forall\ s \in n\ \wedge \alpha \in m$
6:  Select $S_i$, such that $\varsigma_{s}^{i,\alpha} < \varsigma_{y}^{i,\alpha}\ \forall\ y \in n\ \wedge \alpha \in m \wedge y \neq s$.
7:  Allocate $c$ to $S_i$, $iff\ \varsigma_{s}^{i,\alpha} + \Delta t(c) < \tau_{max}^{\varsigma}$ //
8:  If $\varsigma_{s}^{i,\alpha} > \tau_{max}^{\varsigma}\ \forall\ s \in n\ \wedge \alpha \in m$, then
9:      Migrate-task c from $S_i$ to $S_j$, $iff\ \varsigma_{j}^{i,\alpha} + \Delta t(c) < \tau_{max}^{\varsigma}$   // intra-pod migration
10: end if
```

Fig. 5.6. Steps Involved in Low (server) Level.

hotspots and to maintain a uniform range of thermal threshold in every pod. Whenever a new job (a job can have multiple tasks) is arrives to the CPS, the tasks are allocated to the specified server based on the thermal signatures. The HLCC and LLCC are proposed that perform the task allocation, task migration, and traffic redirection, based on the thermal analysis of the node or layer. As depicted in Fig. 5.5, there is LLCC in every pod that has the thermal information of all $S, \alpha,$ and δ. Every node in the CPS is equipped with a heat sensor that measures the temperature and the temperature is updated periodically to the LLCC. In *low (server) level* (Fig. 5.6), the $\varsigma_s^{i,\alpha}$ for all the $S \in Pod(i)$ is measured and observed through sensors periodically. Whenever the value of $\varsigma_s^j, \forall j \in n$ exceeds the maximum threshold temperature of the server (τ_{max}^{ς}), the LLCC migrates some tasks from S^j to S^l, where S^j and $S^l$are connected to the same α.

For the tasks to be migrated successfully to S^l, the constraint $\varsigma_s^l + \Delta T < \tau_{max}^{\varsigma}$, must be satisfied. The ΔT represents the anticipated increase in the temperature as a result of task

```
1:  for $i \leftarrow 1$ to $k$ do
2:      Compute $\xi_n^i \; \forall i \in m \wedge i \in w$
3:      If $\exists \, \xi_n^i \in w$ such that $\xi_n^i > \tau_{max}^{\xi}$, then
4:          Redirect $nt$ from $\delta_i$ to $\delta_j, iff \; \xi_n^j + \Delta t(nt) < \tau_{max}^{\xi}$
5:      end if
6:      If $\exists \, \xi_n^i \in m$ such that $\xi_n^i > \tau_{max}^{\xi}$, then
7:          Migrate-task c from $S_i$ to $S_j, \; iff \; \varsigma_j^{i,\alpha} + \Delta t(c) < \tau_{max}^{\varsigma} \wedge \xi_n^j + \Delta t(nt) <$
    $\tau_{max}^{\xi}$        // intra-pod migration
8:      end if
9:  end for
```
Fig. 5.7. Steps Involved in High (access and aggregate) Level.

migration. If the task migration is not possible amongst the serves under the α_i, then the servers belonging to $\alpha_j, \forall j \in m \wedge j \neq i$ are considered for the migration. The α_i and α_j belongs to the same pod. Moreover, if there is no server available for the migration within the same pod, then inter-pod task migration is performed by enforcing the same constraint.

In *high (access and aggregate) level* (Fig. 5.7), the focus is to avoid the hotspot at access and aggregate layer of the CPS by redirecting the traffic from heavily loaded switches to the lighter ones. Redundant paths are available in the network infrastructure of DC that allows redirection of traffic from one switch to another (Fig. 5.2). The decisions for the redirections are made by LLCC considering the value of ξ_n for every switch. When the value ξ_n^i for α increases from τ_{max}^{ξ}, then task migration is performed by LLCC in the same way was as performed in low level. The reason for the aforesaid is a fact that there is only one path between the access and the servers. However, in case of δ, redundant paths are available. Therefore, whenever the value of $\xi_n^i, \forall i \in w$, exceeds the maximum threshold temperature of the switch (τ_{max}^{ξ}), the LLCC instructs the lower level (server) to redirect the traffic from δ_i to δ_j where both δ belongs to the same pod. The redirection is allowed only if the $\xi_n^j + \Delta T < \tau_{max}^{\xi}$. If the redirection is not possible within the same pod, then inter-pod task migration is performed to take some load off from the switch.

The high level and low level are combined together to form an *intra-pod control*. The goal in intra-pod is to stabilize the temperature of the pod by maintaining the thermal signatures of server, access, and aggregate layer. Local decisions (within the same pod), such as task migration and redirection, are taken by LLCC to stabilize the temperature. However, the inter-pod migrations are performed with the consent of HLCC. Whenever, inter-pod actions have to be performed, the LLCC requests HLCC to provide information about other pods where the tasks can be migrated. Afterwards, the LLCC of the pods can communicate with each other to accomplish the task.

The *inter-pod control* is focused on maintaining the unified thermal threshold value in all the pods. The thermal signatures of nodes in CPS can evolve in order of minutes. Moreover, the power states of servers can change as frequent as milliseconds. Therefore, the threshold temperatures are not absolute values; rather it is a range within which the thermal signatures of the nodes and layers should lie. In inter-pod control, the HLCC periodically monitors the average thermal values of each pod that it receives from sensors. Whenever the thermal signature of the $Pod(i)$ ($\tau_{\rho}^{i} = \S_{s}^{i} + \S_{\partial}^{i} + \S_{g}^{i}$) starts to exceeds the maximum thermal threshold value of the pod (τ_{max}^{ρ}), the HLCC instructs the LLCC of $Pod(i)$to migrate some tasks to $Pod(j), \forall j \in k \wedge$ j $\neq$ i. The migration can be successfully performed only if the $\tau_{\rho}^{i} + \Delta T < \tau_{max}^{\rho}$. Moreover, the server selection and task allocation performed in inter-pod control is same as in low level. The HLCC only has the coarse-grain information of the τ_{ρ}^{i}. The allocations of migrated tasks to servers are performed by LLCC through the use of fine-grained servers information. All of the aforementioned controls work together to make sure that the CPS is operating under a specified temperature range. More detailed information, formal analysis, and behavior of the HLCC and LLCC will be discuss in the next section, using HLPN and Z language.

5.4. Verification Using HLPN, SMT-Lib, and Z3 Solver

Verification is the process of demonstrating the correctness of an underlying system [5.39]. Two parameters are required to verify a model or a system: **(a)** specification and **(b)** properties. In this study, we use bounded model checking [5.40] technique to perform the verification, using SMT-Lib and Z3 solver. In bounded model checking, the description of any system is verified, whether any of the acceptable inputs drives the system into a state where the system always terminates after finite number of steps. The process of bounded model checking involves several tasks: **(a)** *Specification*, the description of the system that states the properties or rules, which must be satisfied by the system to be deemed correct, **(b)** *Modeling*, representation of the system, and **(c)** *Verification*, use of a tool to check whether the specifications is satisfied by the model.

***Definition 2*: (Bounded Model Checking) [5.40].** Formally, given a Kripke Structure $M = (S, S_0 R, L)$ and a k bound, the bounded model checking problem is to find $\{M \vDash_k Ef\}$ where: S is the finite set of states, S_0 is the set of initial states, R is the set of transitions such that $R \subseteq S \times S$, and L is the set of labels.The bounded model checking problem is to find an execution path in M of length k that satisfies a formula f.

Kripke structure is a state transition graph used to represent the behavior of the system [5.41]. In Kripke structure nodes are the set of reachable states of the system, edges represent the transitions, and label functions map nodes to the set of properties hold in the state. Fig. 5.8 shows an example Kripke structure and computational tree where: *S={S1, S2, S3}, S0={S1},*

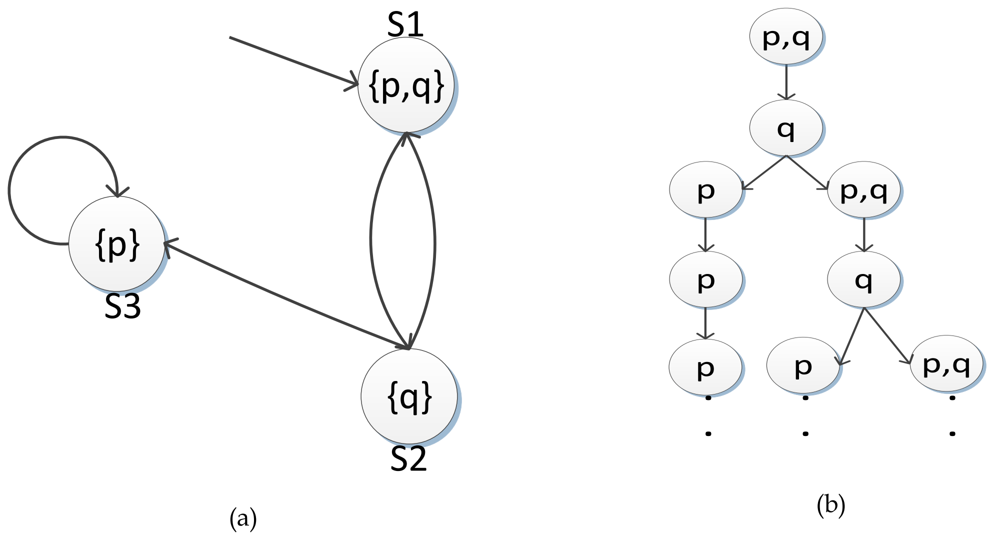


Fig. 5.8. An Example of: (a) Kripke Structure and (b) Computational Tree.

R={(S1,S2),(S2,S1), (S2,S3),(S3,S3)}, p and q are atomic propositions, and $L = \{(S1, \{p, q\}), (S2, \{q\}), (S3, \{p\})\}$.

A path in a Kripke structure can be stated as an infinite sequence of states represented as $\rho = S_1, S_2, S_3, \ldots$ such that for $\forall i \geq 0, (S_i, S_{i+1}) \in R$. The model M may produce a path set $= S_1, S_2, S_1, S_2, S_3, S_3, S_3, \ldots$. To describe the property of a model some formal language, such as CTL*, CTL, or LTL is used. (Readers are encouraged to see [5.42], [5.43] for more details about the CTL*.) For a model to be correct, the states must satisfy the formulas (Definition 2) under a specific bound. The formulas are represented in terms of properties of the systems.

***Definition 3*: (SMT Solver) [5.44].** Given a theory Γ and a formula f, the SMT Solvers perform a check whether f satisfies Γ or not.

To perform the verification of the models using Z3 (an SMT Solver), we unroll the model M and the formula f that provides M_k and f_k, respectively. Moreover, the said parameters are then passed to Z3 to check if $M_k \models_\Gamma f_k$ [5.26]. The solver will perform the verification and provide the results as satisfiable (*sat*) or un-satisfiable (*unsat*). If the answer is *sat*, then the solver will generate a counter example, which depicts the violation of the property or formula f. Moreover, if the answer is *unsat*, then formula or the property f holds in M up to the bound k (in our case k is exec. time).

5.4.1. Modeling HLCC and LLCC Using HLPN

The HLPN model for HLCC and LLCC is shown in Fig. 5.9. The first step towards modeling using HLPN is to identify the required types, Places (P), and mapping (Definition 1). The types and the descriptions are shown in Table 5.1 and the mapping of P to types is depicted in Table 5.2. The description and operation of the controllers are discussed in the previous section and now we can define formulas (pre and post-conditions) to map on transitions. The set T contains the following transitions:

$$T = \{New\,Jobs, Job - Req - F, Job - Req - S, Req - Pod - Ts, Migrate,$$
$$Get - SR, Req - Mg, Req - STs, Job - Alloc, MgR, Sen - Read\}.$$

Table 5.1. Data Types Used in the HLCC and LLCC Model.

Types	**Description**
Task	*A type for the representation of job.*
Res-Mat	*Amount and type of resources available servers.*
Th_S	*A type for the thermal signature (Th. Sig) of the server.*
Th_P	*A type for the Th. Sig of the Pod.*
Th_Ac	*A type for the Th. Sig of the Access Switch.*
Th_Ag	*A type for the Th. Sig of the Aggregate Switch.*
Th_Co	*A type for the Th. Sig of the Core Switch.*
Res	*A type to represent the resources.*
RI	*A type to represent the Routing Information.*
Max_Th_P	*Max. Thermal Threshold (Th. Td) value of the Pod.*
Max_Th_S	*Max. Th. Td value of the Server.*
Max_Th_Ac	*Max. Th. Td of Access Switch.*
Max_Th_Ag	*Max. Th. Td value of Aggregate Switch.*
Max_Th_Co	*Maximum Thermal Threshold value of the core Switch.*
Δt	*Expected thermal dissipation of new task.*

New tokens can only enter the model through the transition $\mathrm{New\,Jobs}$. As seen in Fig. 5.9, no arc is incident on the aforementioned transition, which is why no pre-condition exists and the rules for the transitions can be written as: $\mathrm{R(New\,Jobs)} = \exists \mathrm{j} \in \mathrm{J} \mid \bullet \mathrm{j} = \emptyset$. Whenever the new job arrives, the resource manager checks if the resources required by the job are available or not. The said authentication is performed by the transitions $\mathrm{Job}-\mathrm{Req}-\mathrm{F}$ and $\mathrm{Job}-\mathrm{Req}-\mathrm{S}$, mapped to the following formulas.

$$\boldsymbol{R}(Job-Req-F) = \forall a \in A | \exists J[2] \neq a[2] \wedge \forall\, a[3] \in A \,| \tag{5.16}$$

$$a[3] + \Delta t \geq Max\_Th\_P \wedge A' := A$$

$$\boldsymbol{R}\,(Job-Req-S) = \forall\, a \in A \,|\, \exists\, J[2] \in a[2] \wedge \tag{5.17}$$

$$\forall\, a[3] \in A \,|\, \exists\, a[3] + \Delta t < Max\_Th\_P \wedge$$

$$A' := A \cup \{(J[1], J[2], a[3], a[4])\}$$

Table 5.2. Places Used in the Model of HLCC and LLCC.

Places	**Mappings**
$\varphi(job)$	*ℙ(Task × Res)*
$\varphi\,(RM)$	*ℙ(Task × Res-Mat × Th_P × Th_S)*
$\varphi\,(HL-CC)$	*ℙ(Th_P)*
$\varphi\,(Pod-Sen)$	*ℙ(Th_P)*
$\varphi\,(LL-CC)$	*ℙ(Th_S × Th_Ac × Th_Ag)*
$\varphi\,(AcS-S)$	*ℙ(Th_Ac)*
$\varphi\,(AgS-S)$	*ℙ(Th_Ag)*
$\varphi\,(CN-S)$	*ℙ(Th_S)*
$\varphi\,(Ags)$	*ℙ(RI)*
$\varphi\,(AcS)$	*ℙ(RI)*
$\varphi\,(CoS)$	*ℙ(RI)*
$\varphi\,(CoS-S)$	*ℙ(Th_Co)*
$\varphi\,(CNode)$	*ℙ(Task × Res)*

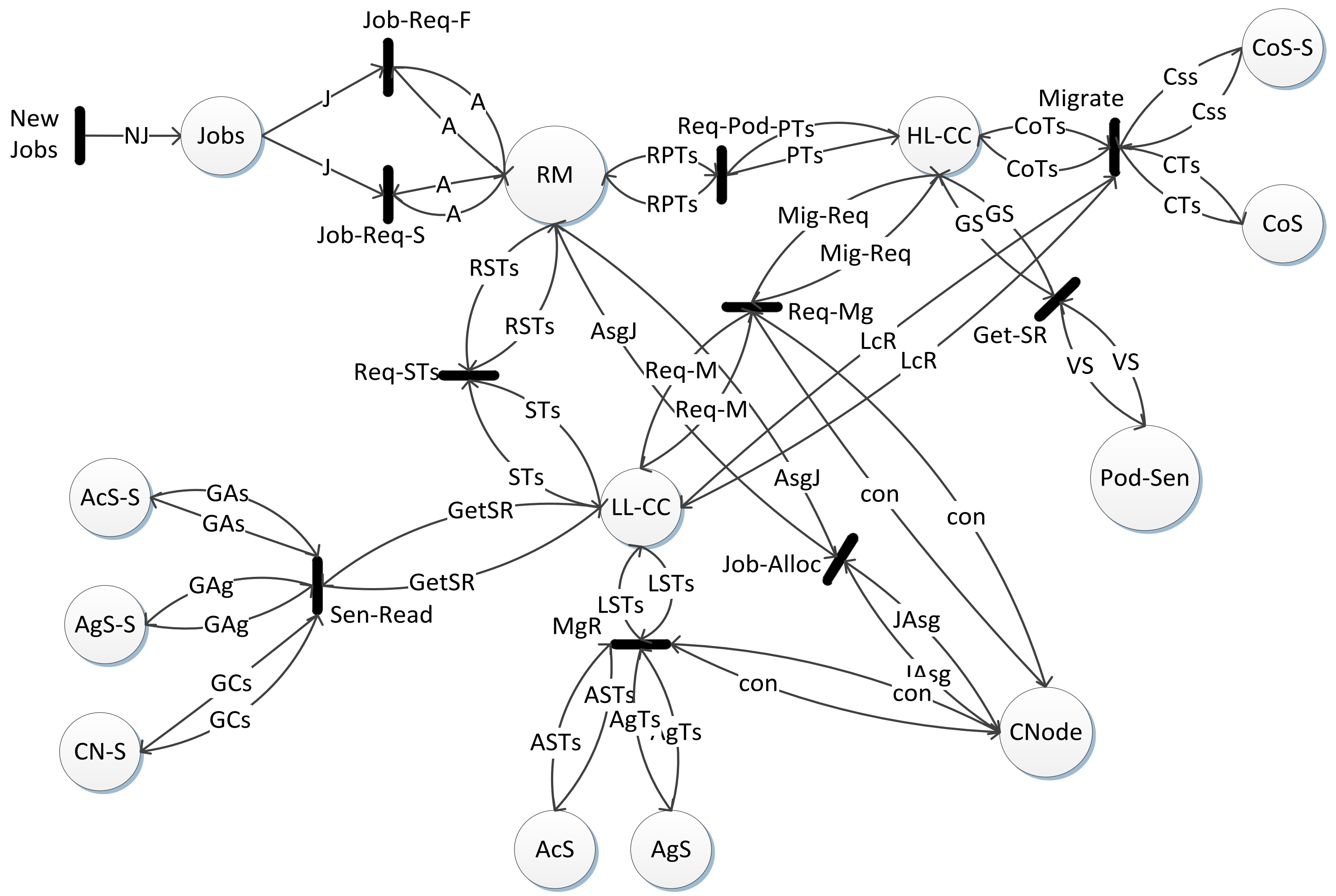


Fig. 5.9. The HLCC and LLCC HLPN Model in DC Environment.

If the resources required by the job are available in the resource matrix of resource manager and the thermal signature of the pod for the selected server is less than the maximum thermal threshold, then the jobs are accepted and are placed in the queue, as shown in (5.17). However, if the resources required by the job are not found, then the job will not be accepted. Moreover, if the cyber portion is running in full capacity, then the job will also be rejected, as in (5.16). The resource manager instructs HLCC and LLCC to provide the list of all the pods and servers that are suitable for the resource allocation. In response, the HLCC provides the thermal information of the pods to resource manager, as shown in (5.18), and LLCC will send the list of all the servers that satisfy the constraint, $\varsigma_s^l + \Delta T < \tau_{max}^{\varsigma}, \forall l \in n \times m \times k$, as in (5.19).

$$\boldsymbol{R}\,(Req - Pod - Ts) = \forall\, rpt[3] \in RPTs, \forall\, pt \in PTs \mid rpt[3] := pt \wedge \tag{5.18}$$
$$RPTs' := rpt[3] \cup \{pt\}$$

$$\boldsymbol{R}\,(Req - STs) = \forall\, rt[4] \in RSTs, \forall\, st[1] \in STs, \forall\, rt[3] \in RSTs \mid \tag{5.19}$$
$$rt[5.4] := \{\forall\, st[1] \bullet st[1] + \Delta t < Max\_Th\_S \wedge st[1] \in rt[3]\}$$
$$RSTs' := RSTs \cup \{(rt[1], rt[2], rt[3], rt[4])\}$$

The HLCC acquires the τ_ρ^i through heat sensors that are placed at each pod (Fig. 5.5). Moreover, the LLCC acquires the ς_s and ξ_n from the heat sensors placed at every node within the pod. The HLCC and LLCC periodically read the values from the sensors, shown in (5.20) and (5.21), respectively. When the resource manager request for the thermal information of the pods and servers, the HLCC and LLCC sends the updated values read from the sensors. The transitions $Get - SR$ and $Sen - Read$ performs the aforementioned readings for HLCC and LLCC, respectively. The rules for the transitions are:

$$\boldsymbol{R}\,(Get - SR) = \forall\, g \in GS, \forall\, v \in VS \mid g := v$$
$$GS' := GS \cup \{(g)\} \tag{5.20}$$

$$\boldsymbol{R}\,(Sen - Read) = \forall\, ac\, \in\, GAs, \forall\, ag\, \in\, GAg, \forall\, gc\, \in\, GCs, \forall\, gsr\, \in\, GetSR\, | \tag{5.21}$$

$$gsr[1] \coloneqq\, gc\, \wedge\, gsr[2] \coloneqq\, ac\, \wedge\, gsr[3]:=\, ag\, \wedge$$

$$GetSR' \coloneqq\, GetSR \cup \{(\, gsr[1], gsr[2], gsr[3])\}$$

$$\boldsymbol{R}(MgR) = \forall act \in ASTs, \forall agt \in AgTs, \forall cot \in\, CoTs, \forall\, cn \in CNode, \tag{5.22}$$

$$\forall\, lst \in LSTs\, |\, lst\left[1_{(i)}\right] \geq Max\_Th\_S \wedge$$

$$\exists\, lst[1_{(j)}]_{\,\forall j \in\, lst[2], j \neq i} + \,\Delta t <\, Max\_Th\_S\, \wedge$$

$$LcMg\,\left(cn_{(i)}, cn_{(j)}\right) \wedge lst\left[2_{(i)}\right] \geq\, Max\_Th\_AC \wedge$$

$$\exists\, lst[2_{(j)}]_{\forall j \in\, lst[2], j \neq i} +\, \Delta t\, <\, Max_{Th_{Ac}} \wedge LcRd\,(lst[2_{(i)}], lst[2_{(j)}])\, \wedge$$

$$lst\left[3_{(i)}\right] \geq\, Max\_Th\_Ag \wedge \exists\, lst[3_{(j)}]_{\,\forall j \in\, lst[3], j \neq i}\, +\, \Delta t\, <\, Max\_Th\_Ag\, \wedge$$

$$LcRd\,(lst[3_{(i)}], lst[3_{(j)}])\, \wedge$$

$$con\left(cn_{(i)}\right)' =\, con\left(cn_{(i)}\right)\left\{\left(\, cn_{(i)}[1], cn_{(i)}[2]\right)\right\}$$

$$\wedge\, con\left(cn_{(j)}\right)' =\, con(cn_{(j)}) \cup \{(\, cn_{(j)}\,[1], cn_{(j)}\,[2])\}$$

If (5.17) is fired, then the job is assigned to the selected server and the resources are allocated to the task, as in (5.22). As stated in the previous section, to maintain a specified thermal temperature at different levels of CPS, the HLCC and LLCC performs task migration and traffic redirections based on the thermal signatures of the nodes. The transition MgR performs the migrations and redirection within the same pod, termed as LcMg and LcRd, respectively.

Whenever the thermal signatures of $S, \alpha,$ and δ are raised more than the specified maximum thermal threshold, the (5.22) is fired. The (5.22) makes local redirection and migration by exploiting the functionalities of LLCC. The inter-pod migration is achieved by the mutual communication of HLCC and LLCC. When migration or redirection is not possible locally, then LLCC requests HLCC to provide the information about the pods where the tasks can be migrated, as depicted in (5.23). Moreover, inter-pod migration is also performed when the thermal signature of γ exceeds the specified maximum thermal threshold, as illustrated in (5.24).

$$\boldsymbol{R}(Req-Mg) = \forall\, rm \in Req-M, \forall\, mr \in Mig-Req, \forall\, cn[1] \in con \mid \tag{5.23}$$

$$(rm[1_{(i)}] + \Delta t > Max\_Th\_S)_{\forall i \in Pod(k)} \wedge$$

$$(rm[2_{(j)}] + \Delta t > Max\_Th\_Ac)_{\forall j \in Pod(k)} \wedge$$

$$(rm[3_{(z)}] + \Delta t > Max\_Th\_Ag)_{\forall z \in Pod(k)} \wedge \exists\, mr(y) > Max\_Th\_P \wedge$$

$$Req-InterPod-Mig\,(cn_{(i)\in Pod(k)}, cn_{(j)\in Pod(x)\neq Pod(k)}) \wedge$$

$$con\big(cn_{(i)}\big)' = con\big(cn_{(i)}\big)\,\{(\,cn_{(i)}[1], cn_{(i)}[2])\} \wedge$$

$$con\big(cn_{(j)}\big)' = con(cn_{(j)}) \cup \{(\,cn_{(j)}\,[1], cn_{(j)}[2])\}$$

$$\boldsymbol{R}(Migrate) = \forall ct \in CoTs, \forall c \in CTs, \forall cs \in Css, \forall\, lr \in LcR \mid \tag{5.24}$$

$$\exists\, c_{(x)} > Max\_Th\_Co \wedge Rd\,(c_{(x)}, c_{(x'),x\neq x'}) \wedge c_{(x')} + \Delta t < Max\_Th\_Co$$

To explain the process of verification, a Kripke structure and an example computational tree of the HLCC and LLCC are formulated and depicted in Fig. 5.10 and Fig. 5.11, respectively. The properties are specified in CTL*. One property to verify the models is that, *there will be no hotspots (overheating) in CPS*. If we closely analyze Fig. 5.10 and Fig. 5.11, we can see that

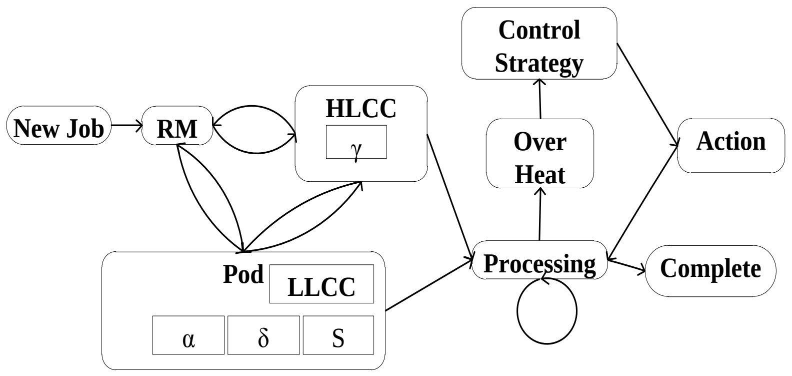


Fig. 5.10. The Kripke Structure of HLCC and LLCC for the Verification.

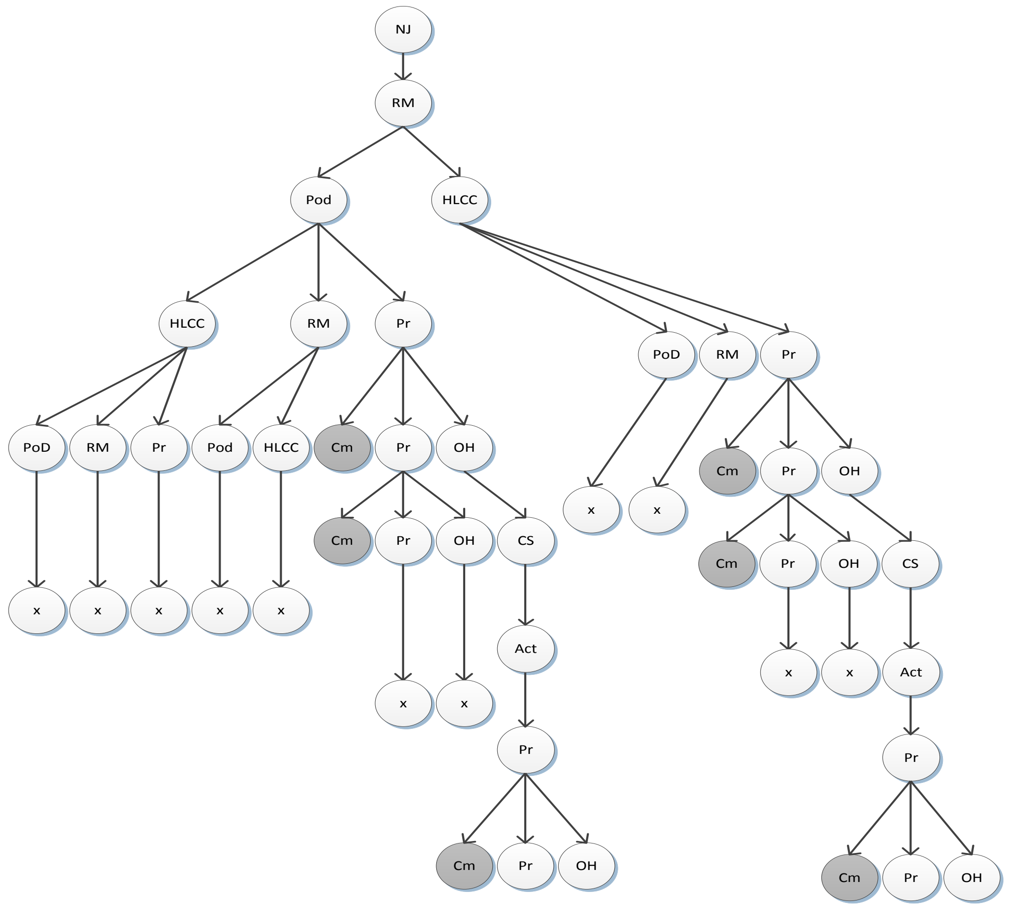


Fig. 5.11. Computational Tree for the Kripke Structure in Fig. 5.10.

whenever the OH (OverHeat) state is reached, the control strategies perform certain actions, such as task migration and redirection (as discussed in above sections) to stabilize the temperature at a desired level, which is Pr (Processing) state. For the models to be correct, the solver should be able to find a terminating state in a model. The failure transitions are considered as a terminator of the models. Moreover, the other terminating state in the models is the last state when the jobs are successfully completed. The state Cm (Complete) in Fig. 5.11 is reachable from every path of the tree, stating that the model will terminate after certain number of iterations. The states labeled with "x" represents, that from the point forward the tree will repeat the predecessor. We have specified the properties of the control strategies in a similar passion and verified whether the properties are satisfied by the models.

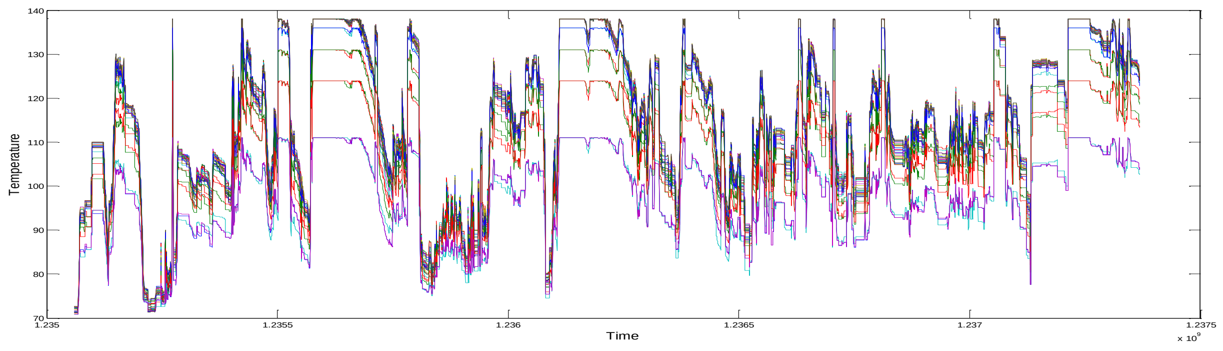
5.5. Results and Discussion

To demonstrate the capacity of our work, we simulate the proposed strategies on a real data center workload obtained from the Center of Computational Research (CCR), State University of New York at Buffalo. All jobs submitted to the CCR are logged for a period of a month. The jobs and the logs from the CCR dataset are used as an input for our simulation of the proposed thermal aware strategy. The dataset had 22,700 jobs (127,000 tasks) recorded in one month of a time. Moreover, we also evaluate the proposed TACS by comparing with a classical First Come First Serve (FCFS), Genetic Algorithm (GA) based thermal aware scheduling [5.23], and Thermal Aware Task Allocation [5.18] approaches.

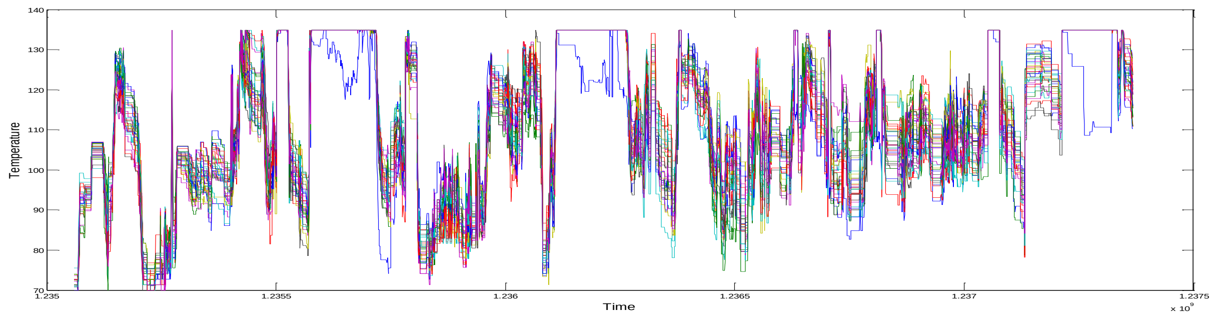
We perform the comparison among the mentioned strategies based on the CCR dataset. Before going deeper into the details of the comparison, we first briefly discuss the existing approaches. The FCFS (sometimes referred as first-in, first-out) is possibly the most straightforward scheduling approach. The jobs are submitted to the scheduler, which dispatches the jobs based on the order of the jobs received.

The FCFS policy is intuitively fair, allowing the jobs that are submitted first to execute first. However, the policy is not preemptive and long running jobs can cause delay to other following jobs. The approach in [5.23] follows the steps of GA. The first step is to construct a set of feasible solutions, which is the task allocation to the servers. Then, the selected solution is mutated (randomly interchange the task allocations within the solution) and mated (randomly select pairs of solution and exchange the subset of two task assignment to get two new solutions). The fitness function, which checks the highest inlet temperature of the selected assignment, is applied to all of the solutions that are formed as a result of mating and mutation, including the original solution. Finally, the solution having the lowest inlet temperature value from the set of highest inlet temperature values, obtained as a result of fitness function, is selected as a final solution. The last approach is TASA proposed in [5.18], which is based on the theory of coolest inlet that perform the assignment of hottest jobs to the coolest servers. The TASA algorithm sorts the servers in the increasing order of the temperatures, such that the coolest server is first in the order. The jobs are sorted in a similar way but in the reverse order, such that the hottest job is first in the order. The hottest job is assigned to the coolest server and the thermal map of all the servers is updated. The same process is repeated until the last job is allocated to the server.

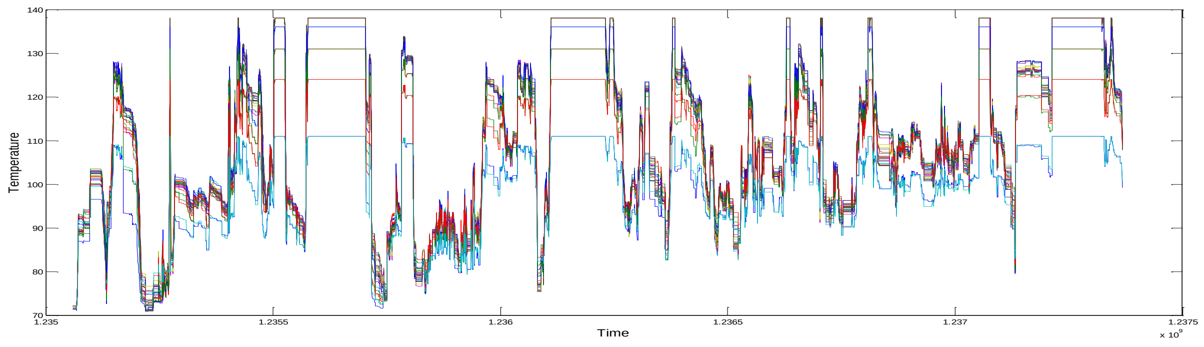
Fig. 5.12 depicts the average thermal signatures of the servers over the period of time, when the scheduling approaches are used. The epoch time stamp and average thermal signature of the servers at that particular time are plotted on x-axis and y-axis in Fig. 5.12. There were 33 pods in the DC and each pod has 32 servers. The thermal readings were taken after every 10 minutes. It can be observed from the Fig. 5.12 that the spread or the difference between the temperatures of the servers in the trend line of Fig. 5.12(a), (b), (c) is very wide at many time stamps. The aforesaid, identify the situation when the average temperature of some servers is lower than the rest of the servers in the DC. Particularly, at time stamps 1.2357E+9, 1.2362E+9, and 1.2372E+9 in Fig. 5.12 (a), (b), (c), the thermal signatures of some servers are very low as compared to the rest, which shows the probable presence of the hotspots in DC. The possible reason for the occurrence of the hotspots in Fig. 5.12(a) is the static assignment of tasks without considering the thermal status of the server that possibly creates a scenario when hot jobs are assigned to hot servers and cold jobs are assigned to cold ones.



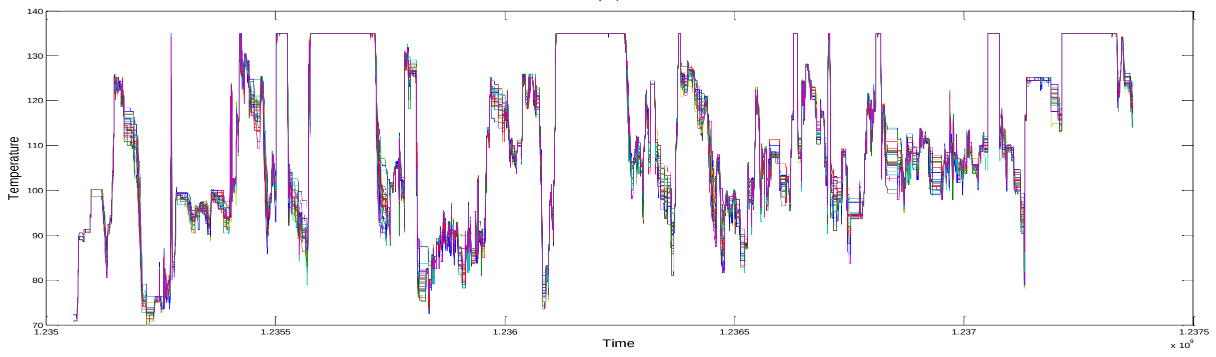
(a)



(b)



(c)

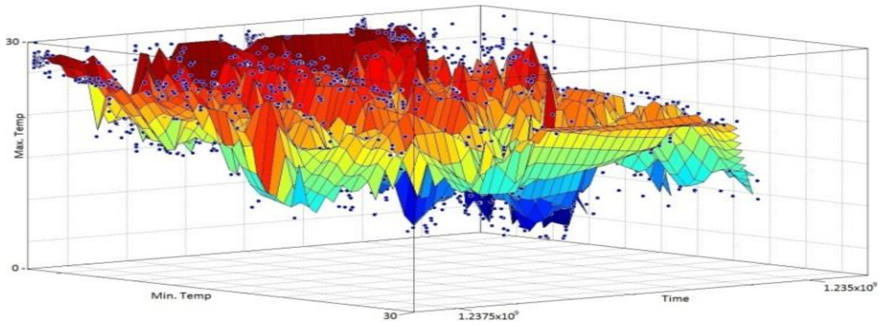


(d)

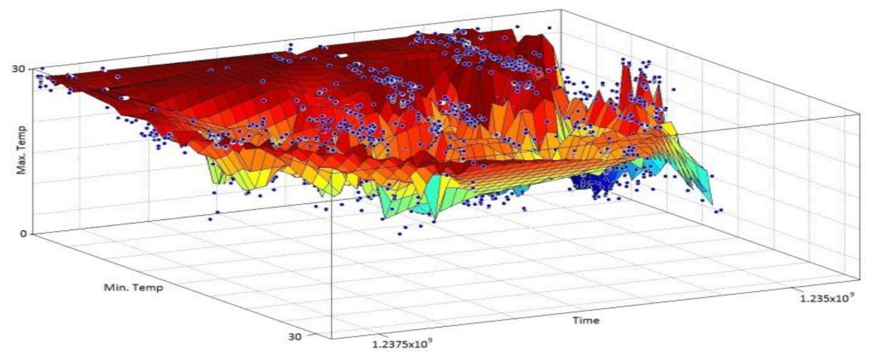
Fig. 5.12. Comparison of Average Thermal Signatures of the Pods Using: (a) FCFS, (b) GA-based, (c) TASA, and (d) TACS.

In Fig. 5.12(b), the reason for the imbalance thermal signatures is the random nature of the GA based approach. The selection of the feasible solution, the mutation, and the mating process, all are based on randomization. If the same set of pods and servers are selected in the solutions most of the time, then the fitness function performed on the selected solution will not provide any important information that will avoid the occurrence of the hotspots. Similarly, there is also a possibility that the number of tasks allocated to few pods and servers are relatively low as compared to the rest of the pods and servers in the DC. The aforementioned possibilities will allow some servers to have high thermal signatures while others have low thermal signatures, which will ultimately cause the hotspot in the DC. In Fig. 5.12(c), which is better than (a) and (b), still has low thermal signatures of some servers as compared to the rest, which results in the occurrence of hotspots. The reason for the aforesaid is that the hottest tasks are allocated to the coolest servers regardless of the overall thermal temperature of the pod and the recirculation effect that can cause the hotspots.

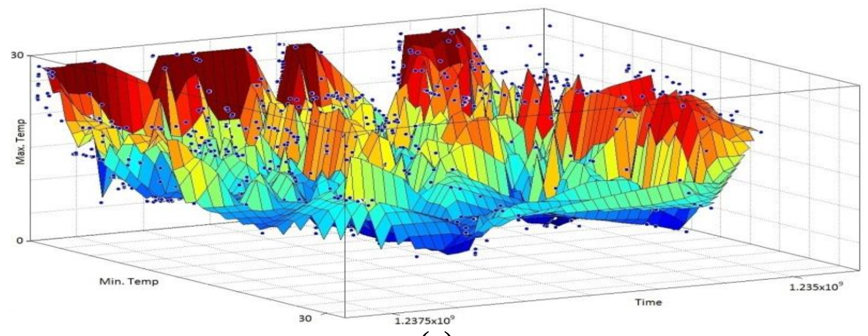
In TACS, as shown in Fig. 5.12(d), the differences of the temperatures amongst the servers are low and there are no hotspots. As stated in Sections 5 and 6, the selection of the pods and servers to allocate the task is based on the thermal signatures. Moreover, the HLCC and LLCC periodically monitor the thermal signatures of the pods and servers, and perform task migration or redirection to maintain unified range of temperatures in the pods. Therefore, the trend of thermal signatures followed in Fig. 5.12(d) is more congested and unified as compared to the trend followed in rest of the approaches. We plot the average difference between the hottest and coolest servers over the period of time (as shown in Fig. 5.13 and 14). The larger and more frequent the differences are, the higher the thermal imbalance will be. We can see that the differences in TACS (d in Fig. 5.13) are very low and less frequent as compared to the other



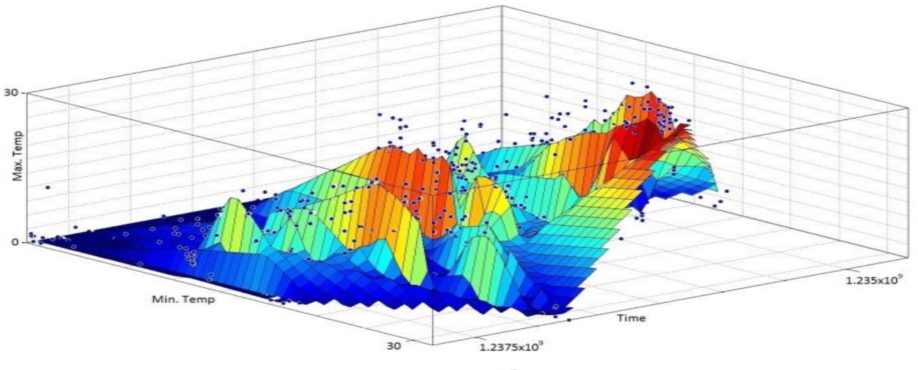
(a)



(b)



(c)



(d)

Fig. 5.13. Comparison of Average Thermal Signature Difference between the Highest and Lowest Servers Using: (a) FCFS, (b) GA-based, (c) TASA, and (d) TACS.

approaches that indicate the thermal balance achieved by using TACS. However, the other approaches have high differences and are occurring frequently, which indicates the thermal imbalance and occurrence of the hotspots.

As stated in previous sections, we also perform the verification of the strategies using SMT-Lib and Z3 solver. To verify, the HLPN models are first translated into SMT. Moreover, the properties are also specified in SMT. Then, the models along with the properties are provided to the Z3 solver, which checks if the properties are satisfied by the models or not. It is noteworthy, that the goal of the verification is to demonstrate the correctness of the models, based on the desirable properties, such as the presence of the hotspots. The results in Fig. 5.15 depict the time taken by the Z3 solver to check the satisfiability of the models, based on the stated property. The property we verify is that, there must be no hotspots in the DC after the task allocation is complete. To accommodate the random behavior of GA based scheduling, we perform the verification of the strategies iteratively for different number of jobs, varying from 10 to 100 jobs, as shown in Fig. 5.15. The verification results matches with the simulation results and no hotspots were identified by the Z3 solver when the proposed TACS was used. However,

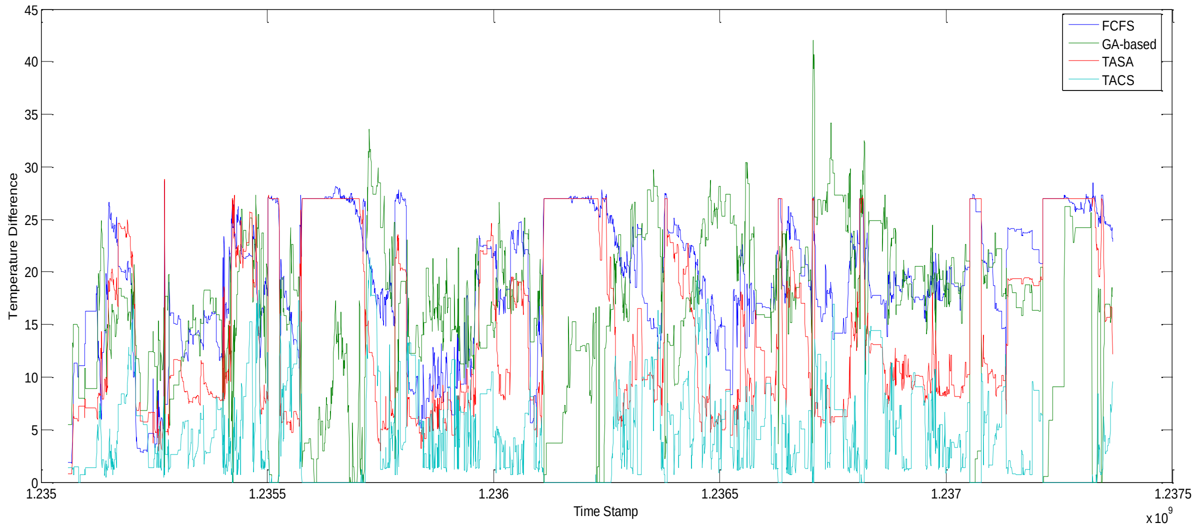


Fig. 5.14. Average Thermal Signature Difference between the Highest and Lowest Servers.

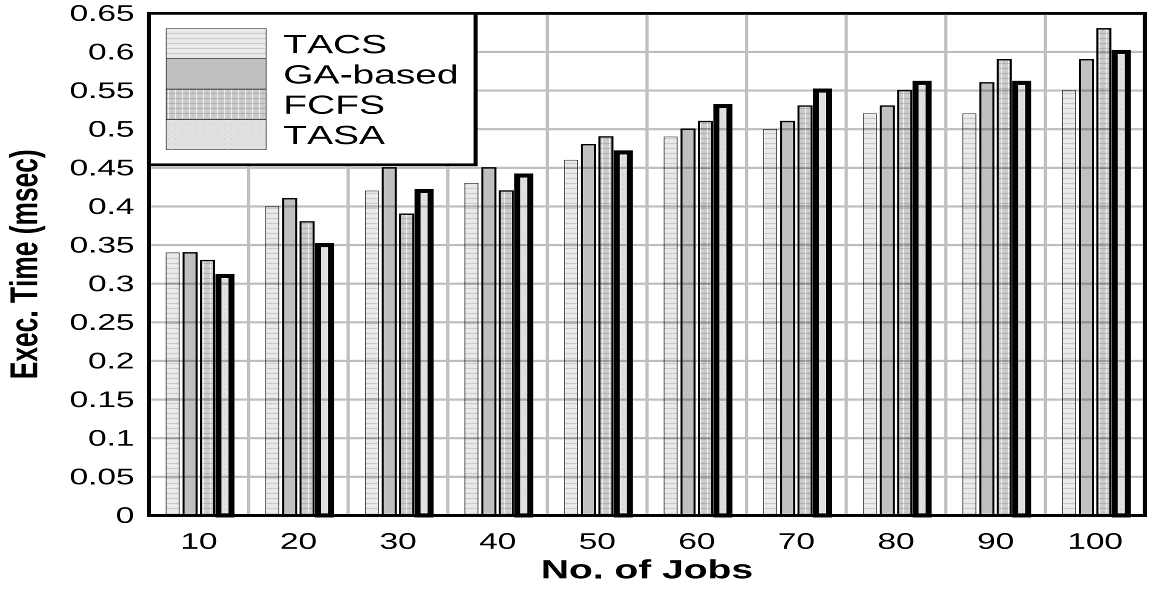


Fig. 5.15. Verification Time Comparison of the Approaches.

hotspots were identified by the solver for the other scheduling approaches at different no. of jobs, as shown in Table 5.3.

Table 5.3. Verification Outcomes of Scheduling Approaches.

# of Jobs	***FCFS***	***GA-based***	***TASA***	***TACS***
10	Unsat	Unsat	Unsat	Unsat
20	Unsat	Unsat	Unsat	Unsat
30	Unsat	Unsat	Unsat	Unsat
40	Unsat	Unsat	Unsat	Unsat
50	Unsat	Sat	Unsat	Unsat
60	Sat	Sat	Unsat	Unsat
70	Sat	Sat	Unsat	Unsat
80	Sat	Unsat	Sat	Unsat
90	Sat	Unsat	Sat	Unsat
100	Unsat	Sat	Unsat	Unsat

We used bounded model checking technique for the verification and in our case, the execution time serve as a bound over the verification models. As stated in Section 6, the solver returns “sat” if the stated assertion is not true, which means that the property is violated. If the property is met by the model, then the solver will return “unsat”, which shows that solver is unable to find the values for which the property is not true. The simulation and verification results reveal that our strategy is consistent and provides better results as compared to the other scheduling approaches. The occurrence of the hotspots may cause servers to throttle down, increasing the possibility of failure. We reduce the possibility of hotspots in our strategy through strategic decisions performed by HLCC and LLCC based on the thermal signatures of the components.

5.6. References

[5.1] R. Buyya, S.Y. Chee, and S. Venugopal, “Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities,” IEEE HPCC, pp. 5-13, 2008.

[5.2] S. U. R. Malik, S. U. Khan, and S. K. Srinivasan, "Modeling and Analysis of State-of-the-art VM-based Cloud Management Platforms," IEEE Transactions on Cloud Computing, pp. 50-63, 2013.

[5.5.3] J. Hamilton, “Cost of power in large-scale data centers,” http://perspectives.mvdirona.com, Nov. 2008.

[5.5.4] U.S. Environmental Protection Agency (EPA). Report to congress on server and data center energy efficiency, public law 109-431, Aug. 2007.

[5.5.5] L. Wang, S. U. Khan, and J. Dayal, "Thermal Aware Workload Placement with Task-Temperature Profiles in a Data Center," Journal of Supercomputing, vol. 61, no. 3, pp. 780-803.

[5.5.6] J. Shuja, S. A. Madani, K. Bilal, K. Hayat, S. U. Khan, and S. Sarwar, "Energy-Efficient Data Centers," Computing, vol. 94, no. 12, 2012, pp. 973-994.

[5.5.7] J. Moore, J. Chase, P. Ranganathan, and R. Sharma, “Making scheduling "cool": temperature-aware workload placement in data centers,” In USENIX, pp. 61-75, 2005.

[5.5.8] L. Ramos and R. Bianchini, “C-oracle: predictive thermal management for data centers,” Symposium on High Performance Computer Architecture, pp. 111–122, 2008.

[5.5.9] L. Parolini, B. Sinopoli, B. Krogh, and W. Zhikui, "A Cyber–Physical Systems Approach to Data Center Modeling and Control for Energy Efficiency," Proceedings of the IEEE, vol. 100, no. 1, , 2012, pp. 254,268.

[5.5.10] L. Parolini, N. Toliaz, B. Sinopoli, and B. H. Krogh, “A Cyber-Physical Systems approach to energy management in data centers,” Conference on Cyber-Physical Systems, 2010.

[5.5.11] Y. Cho and N. Chang, “Energy-aware clock-frequency assignment in microprocessors and memory devices for dynamic voltage scaling,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 26, no. 6, 2007, pp. 1030–1040.

[5.5.12] H. Aydin and D. Zhu, “Reliability-aware energy management for periodic real-time tasks,” IEEE Transactions on Computers, vol. 58, no. 10, 2009, pp. 1382–1397.

[5.5.13] P. Choudhary and D. Marculescu, “Power management of voltage/frequency island-based systems using hardware-based methods,” IEEE Transactions on VLSI Systems, vol. 17, no. 3, 2009.

[5.5.14] A. Varma, B. Ganesh, M. Sen, S. Choudhury, L. Srinivasan, and B. Jacob, “A control-theoretic approach to dynamic voltage scheduling,” International CCASE, pp. 255–266, Oct. 2003.

[5.15] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis, “Power management of datacenter workloads using per-core power gating,” Computer Architecture Letters, 2009, vol. 8, no. 2, pp. 48–51.

[5.16] Z. Jian-Hui and Y. Chun-Xin, “Design and simulation of the cpu fan and heat sinks,” IEEE Transactions on Components and Packaging Technologies, vol. 31, no. 4, 2008, pp. 890–903.

[5.17] A. Mutapcic, S. Boyd, S. Murali, D. Atienza, G. Micheli, and R. Gupta, “Processor speed control with thermal constraints,” IEEE Transactions on Circuits and Systems, vol. 56, no. 9, pp. 1994–2008.

[5.18] L. Wang, V. Laszewski, G. Dayal, J. He, X. Younge, and T. R. Furlani, “Towards thermal aware workload scheduling in a data center,” International Symposium on Pervasive Systems, Algorithms, and Networks, pp. 116-122, 2009.

[5.19] J. Moore, J. Chase, and P. Ranganathan, “Weatherman: Automated, online and predictive thermal mapping and management for data centers,” IEEE ICAC, pp. 155-164, 2006.

[5.20] N. Tolia, Z. Wang, P. Ranganathan, C. Bash, M. Marwah, and X. Zhu, “Unified power and cooling management in server enclosures,” in InterPACK, pp. 721–730, 2009.

[5.21] C. Bash, C. Patel, and R. Sharma, “Dynamic thermal management of air cooled data centers,” Thermal and Thermomechanical Phenomena in Electronics Systems, pp. 445–452, 2006.

[5.22] L. Rao, X. Liu, L. Xie, and W. Liu, “Minimizing electricity cost: Optimization of distributed internet data centers in a multi-electricitymarket environment,” International Conference on Computer Communications (INFOCOM), pp. 1–9, 2010.

[5.23] Q. Tang, S. Gupta, and G. Varsamopoulos, “Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach,” IEEE Transactions on Parallel and Distributed Systems, vol. 19, no. 11, 2008, pp. 1458–1472.

[5.24] M. Anderson, M. Buehner, P. Young, D. Hittle, C. Anderson, J. Tu, and D. Hodgson, “MIMO robust control for HVAC systems,” IEEE Transactions on Control Systems Technology, vol. 16, no. 3, 2008, pp. 475– 483.

[5.25] M. Toulouse, G. Doljac, V. Carey, and C. Bash, “Exploration of a potential-flow-based compact model of air-flow transport in data centers,” American Society Of Mechanical Engineers ASME Conference, pp. 41–50, 2009.

[5.26] M. K. Ganai and A. Gupta, “Accelerating high-level bounded model checking,” in ICCAD, 2006, pp. 794–801.

[5.27] Q. Tang, T. Mukherjee, S.K.S. Gupta, and P. Cayton, “Sensor-
Based Fast Thermal Evaluation Model for Energy Efficient High-Performance Datacenters,” ICISIP, Dec. 2006.

[5.28] T. Murata, “Petri Nets: Properties, Analysis and Applications,” Proc. IEEE, vol. 77, no. 4, 1989, pp. 541-580.

[5.29] Lectures on Petri Nets I: Basic Models, Lecture Notes in Computer Science, vol. 1491, W. Reisig and G. Rozenberg, eds., Berlin: Springer-Verlag, 1998.

[5.30] J. Desel and J. Esparza, “Free Choice Petri Nets,” Cambridge Tracts in Theoretical Computer Science, vol. 40, Cambridge, UK: Cambridge Univ. Press, 1995.

[5.31] K. Bilal, M. Manzano, S. U. Khan, E. Calle, K. Li, and A. Y. Zomaya, "On the Characterization of the Structural Robustness of Data Center Networks," IEEE Transactions on Cloud Computing, vol. 1, no. 1, pp. 64-77, 2013.

[5.32] N. En and N. Srensson, “An extensible SAT-solver,” Lecture Notes in Computer Science, vol. 2919, 2003, pp. 502-518.

[5.33] C P. Gomes, H. Kautz, A. Sabharwal, and B. Selman, “Satisfiability solvers,” In Handbook of Knowledge Representation, 2007.

[5.34] M. Frade and J. S. Pinto, “Verification conditions for source-level imperative programs,” Technical Report DI-CCTC-08-01, University of Minho, 2008.

[5.35] SMT-Lib http://smtlib.cs.uiowa.edu/, accessed Jan. 2013.

[5.36] S. U. R. Malik, S. K. Srinivasan, S. U. Khan, and L. Wang, “A Methodology for OSPF Routing Protocol Verification,” Conference on Scalable Computing and Communications, Dec. 2012.

[5.37] C. Barrett, A. Stump, and C. Tinelli, “The SMT-LIB Standard: Version 2.0,” 8th International Workshop on Satisfiability Modulo Theories, 2010.

[5.38] L. de Moura and N. Bjorner, “Z3: An efficient SMT solver,” Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2008.

[5.39] S. Nakajima, “Model-checking Verification for Reliable Web Service,” OOWS, 2002.

[5.40] A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu, “Bounded model checking”, Advances in Computers, vol. 58, Academic press, 2003.

[5.41] Y. Quemener and T. Jeron, “Model Checking of CL on infinite kripke structures defined by simple grammers,” Technical Report RR-2563, INRIA, France, 1995.

[5.42] M. Maidl, “The common fragment of CTL and LTL,” Symposium on Foundations of Computer Science, pp. 643–652, 2000.

[5.43] K. Bilal, S. U. R. Malik, O. Khalid, A. Hameed, E. Alvarez, V. Wijaysekara, R. Irfan, S. Shrestha, D. Dwivedy, M. Ali, U. S. Khan, A. Abbas, N. Jalil, and S. U. Khan, "A Taxonomy and Survey on Green Data Center Networks," FGCS. (Forthcoming.)

[5.44] L. Cordeiro, B. Fischer, and J. Marques-Silva, “SMT-based bounded model checking for embedded ANSI-C software,” ASE, pp. 137–148, 2009.

6. A METHODOLOGY FOR OSPF ROUTING PROTOCOL VERIFICATION

This paper is presented in *International Conference on Scalable Computing and Communications (ScalCom),* Changzhou, China, December 2012. The authors of the paper are Saif U. R. Malik, Sudarshan K. Srinivasan, Samee U. Khan, and Lizhe Wang.

6.1. Introduction

Data intensive systems, such as data centers, have a real need for tens to hundreds of Gbps of bandwidth and deterministic Quality of Service (QoS), which is satisfied by thousands of servers interconnected together. Data Centers (DC) gained a great popularity for the provision of computing resources [6.3]. Amazon, Google, IBM, Facebook, and Microsoft have started to establish data centers that host cloud computing applications in geographically distributed locations [6.2]. DCs contains a pool of computing resources to host applications and store data, connected together using communication medium, such as fiber optics. The performance and stability of the network depends on the performance of the routing mechanisms implemented within the architecture [6.4]. Routing protocol plays an important role towards the performance realization of large scale networks. Therefore, it is compulsory to verify the working of the routing protocol to ensure reliable communication amongst the systems in the network.

Open Shortest Path First (OSPF) is an adaptive routing protocol that is used for Internet Protocol (IP) networks to distribute routing information within a single Autonomous System (AS) [6.5]. OSPF divides the network into areas, as shown in Fig. 6.1 [6.1]. Each area consists of one or more segments. A segment constitutes the set of routers connected via a common communication channel, such as Ethernet. With the rapidly increasing and changing demands of

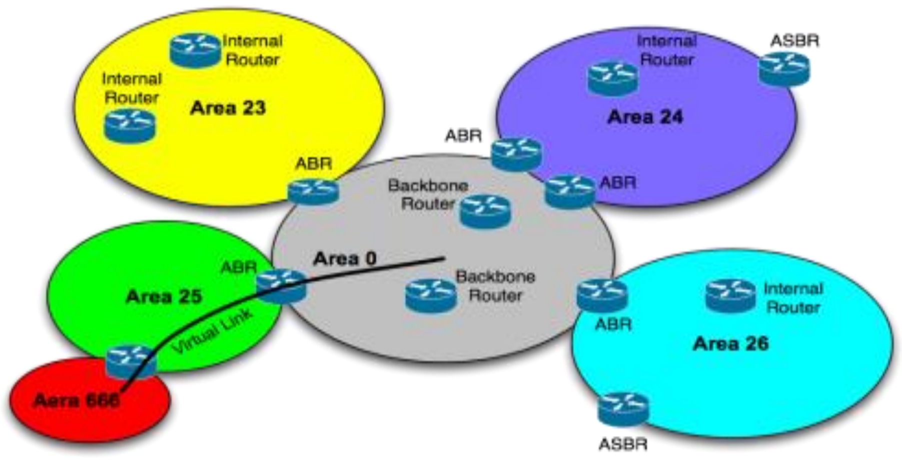


Fig. 6.1. OSPF Areas and Routers.

QoS, modern routing domain, such as DCs need to maintain a very high level of service availability. Therefore, OSPF should attain fast convergence in response of topology changes, to meet the demands of modern systems. Moreover, to avoid loss of messages, the information flowing within the data center must be routed correctly by the OSPF. A slight misinformation can cause huge packets loss depending on the size of the network. In the aforementioned aspect, we have verified OSPF protocol using SMT-Lib and Z3 Solver.

6.1.1. SMT-Lib and Z3 Solver

Satisfiability Modulo Theories (SMT) is an area of automated deduction for checking the satisfiability of formulas with respect to some logical theory of interest [6.18]. SMT has been used in many fields including deductive software verification. Moreover, recent applications of computer science including planning, model checking, and automated test generation finding, also considers SMT as an important verification tool [6.19]. The solver can be distinguished amongst the features they provide, such as, underlying logic (example first order or temporal), background theories, input formulas, and interface. The details about the features can be found in [6.30]. Multiple solvers are available that supports SMT-LIB, such as Beaver, Boolector, CVC4, MathSAT5, Z3, and OpenSMT [6.19].

We used Z3 solver in our study, which is a high performance theorem prover developed at Microsoft Research. Z3 is an automated satisfiability checker. Moreover, Z3 also checks whether the set of formulas are satisfiable in the built-in theories of SMT-LIB. Readers are encouraged to see [6.20], for the detailed information about the working and commands of Z3 solver. In this paper, we propose a novel method to verify the OSPF routing protocol that incorporate Designated Routers (DRs). The proposed method uses delay information of the router as a property to verify the protocol. We used the delay information to identify the occurrence of events as an update is received by the corresponding DRs. The proposed method can scale up the verification process by reducing the size of state space and limiting it to a single parameter. We used Satisfiability Modulo Theory (SMT-LIB) library and Z3 solver as a tool for the verification purpose. Moreover, BRITE [6.9] topology generator is used to generate the topologies that represents characteristics similar to those of Internet. There are four steps involved in our verification process: **(a)** we have simulated the detailed implementation of OSPF protocol based on the specifications available in [6.12] on a small scale network, **(b)** we modeled the system and specified the properties, **(c)** the model and properties in SMT-LIB are given to Z3-Solver for model checking, and **(d)** execution and generation of results.

6.2. OSPF Routing Protocol

The OSPF is a link-state routing protocol [6.6]. The link state is the description of the interface of the router (IP address of the interface, mask, type of network, routers connected to) and the relationship to other routers. OSPF constructs a topological map of the entire network by gathering the link state information from available routers [6.1]. Unlike other routing protocols, such as Routing Information Protocol (RIP) that uses Bellman-ford vector based algorithms, OSPF introduces new concepts, such as areas, Variable Length Subnet Mask (VLSM), and route summarization [6.14].

To decrease the intra-area convergence time, a router amongst the routers is selected as a DR in OSPF. All other routers on a segment communicate only to the DR, which cuts the information flow cost from $O(n^2)$ to $O(n)$ (instead of sending update to every router on a segment the update is sent to a DR and then that DR will flood the update to all of the other routers) [6.6]. Table 6.1 illustrates the types of routers used in OSPF. The type of router is identified based on the router interface and link states. Do not confuse DR with OSPF router types. A router can have some interfaces that are designated, which makes a router DR. Moreover, different types of routers generate different Link State Advertisements (LSA), which is a way to communicate the routing topology to other router in and outside an area. Table 6.2 depicts some of the basic LSAs supported by the routers. Note, that there are other LSA types 6-11, whose information can be found in [6.15].

Table 6.1. The OSPF Routers.

Router Type	Description
Internal	Router that has all the interfaces in single area
Backbone	Router that has at least one interface in backbone area
Area Border	Router having at least one interface in backbone area and another in non-backbone area
Autonomous System Boundary	Router performing route injection from other source (RIP, EIGRP) into OSPF domain.

Table 6.2. The OSPF Link States and Associated Routers.

LSA Type	Description	Associated Router	Scope
1	Describes directly attached link to a router within an area.	All routers	Intra area
2	Describes the number of routers attached in a segment. Gives information about the subnet mask of a segment	DR	Intra area
3	Describes destinations outside an area to flood information from one area to another.	ABR	Inter area
4	Describes a route and information to an ASBR outside the area.	ABR	Inter area
5	Defines routes to destinations external to OSPF domain.	ASBR	Inter area

6.3. Problem Formulation

The problem formulation is taken from our previous work in [6.31]. However, the formulation is modeled accordingly to accommodate the verification aspect of the OSPF protocol. Consider a network composed of N routers. Let R_i be the i^{th} router, where $1 \leq i \leq N$. A link between two routers R_i and R_j (if it exists) has a communication cost *(del)* that represents the minimum time for transferring message from R_i to R_j, which can be represented by the following expression [6.29, 6.31]:

$$del(R_i, R_j) = \frac{D(R_i, R_j)}{v} + \frac{s}{\beta_{ij}}, \qquad (6.1)$$

where $\mathrm{D}(\mathrm{R_i}, \mathrm{R_j})$ is the physical distance between $\mathrm{R_i}$ and $\mathrm{R_j}$, v is the propagation delay of the medium (optical fiber in our case), s is the size of the message in kilobytes, and β_{ij} is the available bandwidth between $\mathrm{R_i}$ and $\mathrm{R_j}$. If the routers are not directly connected, then the communication cost is the sum of the cost of all links in the shortest path from $\mathrm{R_i}$ to $\mathrm{R_j}$. Without the loss of generality, we assume that $\mathrm{del}(\mathrm{R_i}, \mathrm{R_j}) = \mathrm{del}(\mathrm{R_j}, \mathrm{R_i})$, which is a common assumption in literature [6.29]. Let M be the number of segments within an area and $\mathrm{S^k}$ be the *kth* segment in that area, where $1 \leq \mathrm{k} \leq \mathrm{M}$. Let DS be the set of DRs within an area and ς^{k} is the convergence time (time a router takes to discover the area topology) of $\mathrm{S^k}$. If a failure occurs (could be a link or a router), the routers connected to the failed link or failed router will initiate the updates. Let $\mathrm{R_o^k}$ be a router that initiates an update in response to a failure. Let Rr be the set of all other routers in the area defined as $\mathrm{Rr} = \left(\bigcup_{\mathrm{i}=1}^{\mathrm{N}} \{\mathrm{R_i}\}\right) \backslash \{\{\mathrm{R_o^k}\} \cup \mathrm{DS}\}$.

Suppose $\mathrm{R_o^k}$ gets an update, such as node failure. $\mathrm{R_o^k}$ will update its link state and forward the updated link state to the DR of segment *k* (represented as $\mathrm{d^k}$). The link state is the description of the interface of the router (IP address of the interface, mask, type of network, routers connected to) and the relationship to other routers. The DR will then flood the information to every other router in the segment after receiving the update. The verification of the routing protocol can be done in two aspects: **(a)** content verification (if the link state is being calculated correctly) and **(b)** routing verification (if the information is propagated correctly in a same order). For **(a)**, the Link State Database (LSDB) should be same for all the routers after convergence is achieved, such that $\mathrm{LSDB}(\mathrm{R_i^k}) = \mathrm{LSDB}(\mathrm{R_{i+1}^k}) \ldots = \mathrm{LSDB}(\mathrm{R_n^k}) \forall \mathrm{k} \in \mathrm{S}$. For **(b)**, let $Ж^{\mathrm{k}}$contains the list of router that belongs to segment *k* in ascending order of $\mathrm{del}(\mathrm{d^k}, \mathrm{R_i^k}) \forall \mathrm{k} \in \mathrm{S}: \mathrm{R_i} \in \mathrm{k}$. All the routers in a segment must receive the updates in a same order as listed in $Ж^{\mathrm{k}}$.

Let R_i^k represent a router R_i that belongs to S^k. The time for R_i^k to receive the update $\left(\psi(R_i^k)\right)$ can be calculated as follows [6.31]:

$$\psi(R_i^k) = \begin{cases} DI, & if\ R_i^k = R_o^k \\ \psi_{\forall k \in S: R_i \in k}(d^k) + del(d^k, R_i^k), & if\ R_i^k \in Rr \end{cases} \tag{6.2}$$

where,

$$\psi(d^k) = \psi_{\forall j \in k: k \in S}(R_j^k) + del(R_j^k, d^k). \tag{6.3}$$

Other updates, such as change in bandwidth ($\Delta\beta_{ij}$) are assumed to be local and incur zero update time. Therefore in (6.2), the value of $\psi(R_i^k)$ for $R_i^k = R_o^k$, is DI. The DI of routers is usually four times the "Hello" interval, which is the time between consecutive transmissions of "Hello" packets that are used to indicate the liveliness of nodes. The "Hello" interval is 10 seconds for broadcast and P2P networks, and 30 seconds for all other media [6.2]. The value of $\psi(R_i^k)$ for $R_i^k \in Rr$ is the sum of the time required for d^k to receive updates and the time d^k takes to deliver updates to R_i^k. The value of $\psi(d^k)$ is calculated in (6.3), which is the sum of $\psi(R_i^k)$ (the node sending the update to d^k) and the communication cost between them, which is given as $del(R_j^k, d^k)$. Moreover, (6.2) and (6.3) are used to calculate ς^k based on the following expression [6.31]:

$$\varsigma^k = \max\left(\psi(d^k) + del_{\forall j \in k}(d^k, R_j^k)\right). \tag{6.4}$$

The last router (maximum time taken to receive an update from the corresponding d^k) in S^k that receives the update, determines ς^k. Now, using (2), (3), and (4) the convergence time of an area τ can be calculated as follows:

$$\tau = \max_{\forall k \in S}\left((\varsigma^k)\right) + \psi\left((R_o^k) + del(R_o^k, d^k)\right). \tag{6.5}$$

The maximum ς^k amongst all of the segments plus the time when the update is initiated and reaches to the respective DR determines the value of τ.

6.4. Verification of OSPF Using Proposed Method

Verification is the process to demonstrate the correctness of the underlying system [6.16]. We verify the correctness of OSPF through **(a)** content verification and **(b)** route verification as discussed in problem formulation. Note that our goal is to verify the correctness and not to measure the performance of the protocol. Two parameters are required for the verification of the system, namely specification and properties. We achieved verification through model checking [6.17], using SMT-Lib and Z3 Solver. The detailed description for the possible behavior of the protocol (specification) along with the desirable behavior (properties) of the protocol are modeled in SMT and provided to Z3. Given the aforementioned parameters Z3 can perform a verification of the model. Z3 generates a counter example in case of an error that represents the state or values for which the model is incorrect. If there are no errors, then the model specifications can be fine-tuned until converged to the real system. The proposed method can scale up the verification process by reducing the size of the state space and narrowing it down to a single parameter. In the following section we will discuss content verification and route verification in detail.

6.4.1. Content Verification

The OSPF is a link state protocol and all routers in an area must have the same LSDB in order for the protocol to work correctly [6.1]. We assume that the DR is already being elected and initial LSDB synchronization is already being achieved. In content verification, we analyze the state of LSDB for all routers in an area as an update is generated and propagated by the corresponding DR. For content verification, we have simulated the detailed implementation of

OSPF on multi-access segments having multiple DRs in one area. The system model and the property to verify are generated in SMT and are provided to Z3. The property to verify for content verification is that LSDB should be same for all the routers after convergence is achieved, such that $\mathrm{LSDB}(\mathrm{R}_{\mathrm{i}}^{\mathrm{k}}) = \mathrm{LSDB}(\mathrm{R}_{\mathrm{i+1}}^{\mathrm{k}}) = \cdots = \mathrm{LSDB}(\mathrm{R}_{\mathrm{n}}^{\mathrm{k}}) \forall \mathrm{k} \in \mathrm{S}$. Whenever an update occurs, the router initiating an update generates a LSA. The LSA must be propagated to all the routers in an area to have the same view of the topology and to reach the stable state. The aforementioned is necessary to avoid message loss and for the protocol to work properly.

6.4.2. Route Verification

The LSA generated by the routers in case of updates must be propagated to corresponding DR, and then from DR to all other routers in a segment. We propose the use of delay information of routers for route verification. The delay of routers is calculated using (1) as discussed in problem formulation. The delay information is further utilized to order the events as an update occurs. Maintaining the order of the routers reduces the size of state space while verifying the protocol. If no such information is available, then all scenarios have to be considered during the verification process. Suppose we have a topology as shown in Fig. 6.2 below, with arbitrary delays. The topology has two segments. Segment 1 (S^1) has six routers (R_0, *R1*, *R2, R4, R5*, *R7*) and *R2* is the DR (d^1) of S^1. Segment 2 (S^2) has three routers (R_3, R_4, R_6) and R_3 is the DR (d^2) of S^2. Suppose d^1 receives an update and $\psi(\mathrm{d}^1) = 0$. Then using (2) we can calculate the $\psi(\mathrm{R}_{\mathrm{i}}^1) \forall \mathrm{i} \in \mathrm{S}^1$. R_4 is the connecting router between the two segments. Therefore, the update will be propagated from S^1 to S^2 through R_4. The $\psi(\mathrm{d}^2) = 0.28$ (using (3)).If we want to verify the routing, then we can compare the difference of update time of routers and DR $\left((\psi(\mathrm{R}_{\mathrm{i}}^{\mathrm{k}}) - \psi(\mathrm{d}^{\mathrm{k}}))\forall \mathrm{k} \in \mathrm{S}: \mathrm{R}_{\mathrm{i}} \in \mathrm{k}\right)$ with $Ӝ^{\mathrm{k}}$ to verify the routing.

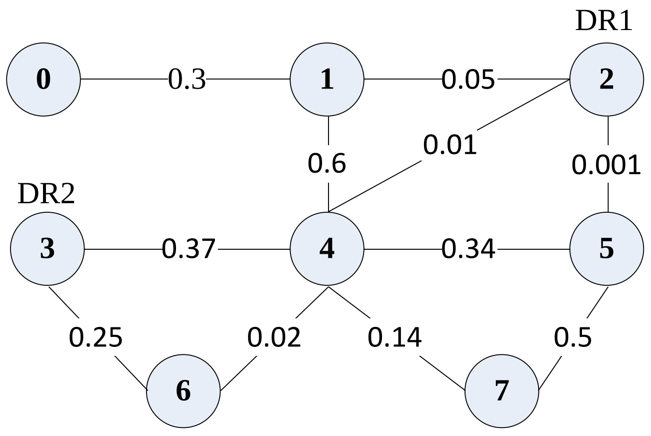


Fig. 6.2. Example Topology and Associated Delays.

We can analyze from Table 6.3 that the values of $\left((\psi\left(R_i^k\right) - \psi\left(d^k\right))\forall k \in S: R_i \in k\right)$ and $Ж^k$ are identical, which indicate that the routing is done correctly and all the routers are receiving the updates in a correct order and time. If the values are not identical, then the protocol may have a problem.

Table 6.3. Comparison of Update Time and Ordered List of Router for Example.

$Ж^1$	$(\psi(R_i^1) - \psi(d^1))$	$Ж^2$	$(\psi(R_i^2) - \psi(d^2))$
0.001 (R5)	0.001-0=0.001	0.25 (R6)	0.53-0.28=0.25
0.01 (R4)	0.01-0=0.01	0.27 (R4)	0.55-0.28=0.27
0.05 (R1)	0.05-0=0.5		
0.15 (R7)	0.15-0=0.15		
0.35 (R0)	0.35-0=0.35		

6.5. Result and Discussion

Performance realization of large scale networks depends highly on the routing protocols. Therefore, it is compulsory to verify the working of the routing protocol to ensure reliable communication amongst the systems in the network. To this end, we have simulated the detailed implementation of OSPF, based on the specifications reported in [6.12] for **(a)** content verification and **(b)** route verification. In OSPF, the routers are usually the Level3 (L3) routers. Therefore, we used optical fiber as a communication medium having propagation delay $v = 300 \times 10^6 \text{miles/sec}$. Ethernet channels have the Maximum Transmit Unit (MTU) of 1500 bytes and in OSPF the fragmentation is usually avoided [6.1]. Therefore, the message size *s* is kept as 1KB, which is neither low nor high and which is typically used in the literature for experimentation (modeling, simulation, and testing). (Readers are encouraged to see the work reported in [6.7] and [6.8] to get and insight into the typical modeling and simulation parameters pertaining to the OSPF modules). Moreover, the bandwidth value of β_{ij} is kept at 100Mbps, as advocated in [6.9, 6.10, 6.11] for the evaluation purposes. The values of $D(R_i, R_j)$ are assigned from within the range of [1-100] km.

Fig. 6.3 depicts the execution time for the content and route verification. For content verification the link state for all the routers in an area must be same. To verify the aforementioned property, we have modeled the simulated system in SMT and generated link

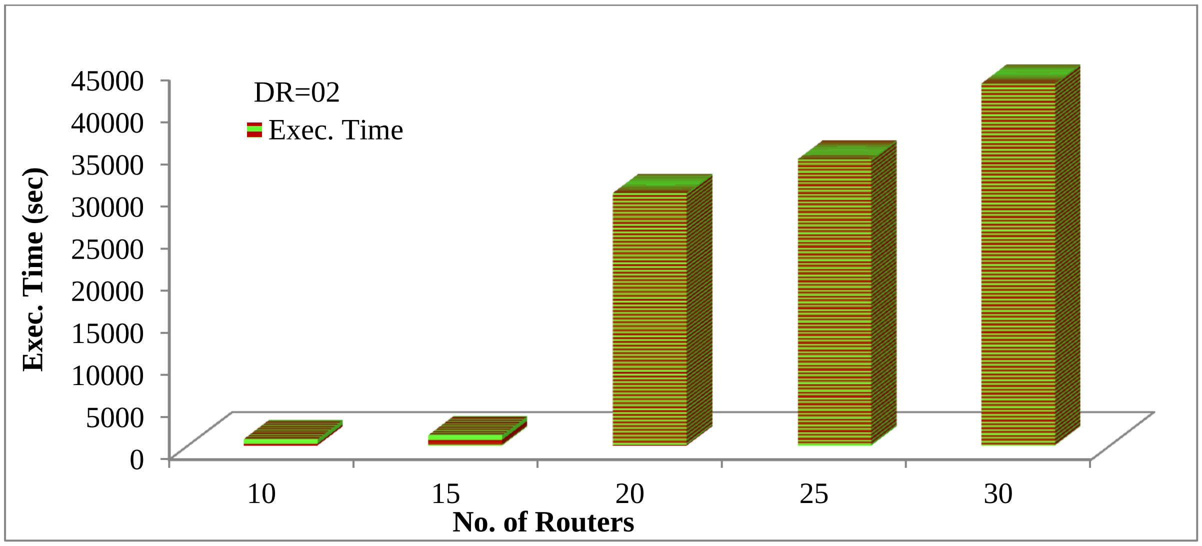


Fig. 6.3. Execution Time for Verification Process.

states for all the routers. When the convergence is achieved, then the link states of all the routers are compared with each other to verify the similarities. For route verification, as discussed in above section, the values of $\left((\psi\left(R_i^k\right) - \psi\left(d^k\right))\forall k \in S: R_i \in k\right)$ and $Ж^k$ must be identical in order for the protocol to work properly. The system model is verified to check if the aforementioned property is satisfied using SMT and Z3 solver. For our implementation using SMT-LIB, we used QF_AUFLIA logic [6.19], which is used for closed quantifier-free linear formulas over the theory of integer arrays extended with free sort and function symbols.

6.6. References

[6.1] J. T. Moy, ‘OSPF; Anatomy of an Internet Routing Protocol’, Addison-Wesley, 1998.

[6.2] D. Abadi, “Data management in the cloud: Limitations and opportunities”, *IEEE Data Engineering, Bulletin*, Vol. 32, No. 1, 2009, pp.3-12.

[6.3] D. Kliazovich, P. Bouvry, and S. U. Khan, "DENS: Data Center Energy-Efficient Network-Aware Scheduling," *ACM/IEEE International Conference on Green Computing and Communications (GreenCom)*, Dec. 2010, pp. 69-75.

[6.4] A. Shaikh, C. Isett, A. Greenberg, M. Roughan, and J. Gottlieb, “A case study of OSPF behavior in a large enterprise network”, *ACM SIGCOMM Internet Measurement Workshop*, France, 2002.

[6.5] A. Caslow, Cisco Certification: Bridges, Routers & Switches for CCIEs. Upper Saddle River, NJ: Prentice Hall PTR, 1998, pp. 373-410.

[6.6] Cisco, ‘OSPF Design Guide’, http://www.cisco.com/en/US/tech/tk365/technologies_white_paper09186a0080094e9e.shtml, accessed 04 May, 2012.

[6.7] I. Krinpayorm, and S. Pattaramalai, “Link Recovery Comparison Between OSPF & EIGRP”, ICICN, 2012, pp. 192-197.

[6.8] B. Wang, J. Zhang, Y. Guo, and W. Chen, “Fast-Converging Distance Vector Routing Mechanism for IP Networks”, *Journal of Networks,* 2010, Vol. 05, No. 9, pp. 1069-1075.

[6.9] HP, “Building Virtualization-Optimized Data Center Networks”, Technical Report. 4AA3-3346ENW, HP, 2011.

[6.10] R. S. Prasad, M. Murray, C. Dovrolis, K. Claffy, “Bandwidth Estimation: Metrics, Measurement Techniques, and Tools”, *IEEE Network*, 2003, Vol. 17, No. 06, pp. 27-35.

[6.11] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. S. E. Ng, M. Kozuch , and M. Ryan, “c-Through: Part-time Optics in Data Centers”, *SIGCOMM*, 2010.

[6.12] Moy, J. (April 1998). RFC 2328 "OSPF Version 2," The Internect Society OSPFv2.

[6.13] G. Retvari, F. Nemeth, R. Chaparadza, and R. Szabo, “OSPF for Implementing Self-adaptive Routing in Autonomic Networks: A Case Study”, *Mid-American Association for Computers in Education (MACE)*, 2009, pp. 72-85.

[6.14] D. Katz, K. Kompella, D. Yeung, “Traffic Engineering (TE) Extensions to OSPF Version 2”, http://tools.ietf.org/rfc/rfc3630.txt, accessed on 02 May, 2012.

[6.15] RFC 2370, the OSPF Opaque LSA Option, IETF Network Working Group 1998.

[6.16] Verification, http://en.wikipedia.org/wiki/Verification

[6.17] P. Wolper. An Introduction to Model Checking, 1995.

http://www.montefiore.ulg.ac.be/~pw/papers/papers.html,

[6.18] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. Satisfiability Modulo Theories. Handbook of Satisfiability, Vol. 185, chapter 26, pp. 825-885. IOS Press, February 2009.

[6.19] SMT-LIB, http://www.smtlib.org/

[6.20] Z3,http://research.microsoft.com/en-us/um/redmond/projects/z3/

[6.21] R. de Renesse and A. Aghvami, "Formal verification of ad-hoc routing protocols using SPIN model checker", *12th IEEE Mediterranean Electro technical Conference,* 2004, pp. 1177–1182.

[6.22] D. Engler and M. Musuvathi, "Static analysis versus software model checking for bug finding", *Verification, Model Checking, and Abstract Interpretation, 5th International Conference,* Lecture Notes in Computer Science, 2004, pp. 191–210.

[6.23] C. Xiong, T. Murata, and J. Tsai, "Modelling and simulation of routing protocol for mobile ad hoc networks using coloured Petri nets", *Workshop on Formal Methods Applied to Defence Systems in Formal Methods in Software Engineering and Defence Systems,* 2002.

[6.24] V. Vishwanath, L. Zuck, J. Leigh, "Specification and verification of LambdaRAM – a wide-area distributed cache for high performance computing" *6th IEEE/ACM Conference on Formal Methods and Models for Codesign (MEMOCODE) 2008,* USA, June 2008.

[6.25]. S. Chiyangwa, M. Kwiatkowska, "A timing analysis of AODV", *Formal Methods for Open Object-Based Distributed Systems: 7th IFIP WG 6.1 International Conference (FMOODS),* (2005).

[6.26] D. Obradovic, Formal Analysis of Routing Protocols. PhD Thesis, University of Pennsylvania (2002).

[6.27] S. Das, D. L. Dill, "Counter-example based predicate discovery in predicate abstraction", *Formal Methods in Computer-Aided Design, Springer-Verlag,* (2002).

[6.28] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An Approach to Universal Topology Generation", *MASCOTS,* Cincinnati, Ohio, 2001.

[6.29] Khan, S. U., and Ahmad, I., “A Pure Nash Equilibrium based Game Theoretical Method for Data Replication across Multiple Servers”, *IEEE TKDE*, 2009, Vol. 21, No. 4, pp. 537-553.

[6.30] C. Barrett, A. Stump, and C. Tinelli, “The SMT-LIB Standard: Version 2.0”, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories*, 2010.

[6.31] S. U. R. Malik, S. K. Srinivasan, and S. U. Khan, "Convergence Time Analysis of Open Shortest Path First Routing Protocol in Internet Scale Networks," *IET Electronics Letters*, vol. 48, no. 19, pp. 1188-1190, 2012.

7. DASCE: DATA SECURITY FOR CLOUD ENVIRONMENT WITH SEMI-TRUSTED THIRD PARTY

This paper is submitted to *IEEE Transactions on Cloud Computing (TCC)* and is in the second round of review. The authors of the paper are Mazhar Ali, Saif U. R. Malik, and Samee U. Khan.

7.1. Introduction

Cloud computing has emerged as a promising computing paradigm and has shown tremendous potential in managing the hardware and software resources located at third-party service providers. On-demand access to the computing resources in a pay-as-you-go manner relieves the customers from building and maintaining complex infrastructures [7.1, 7.13]. Cloud computing presents every computing component as a utility, such as software, platform, and infrastructure. The economy of infrastructure, maintenance, and flexibility makes cloud computing attractive for organizations and individual customers [7.32]. Despite benefits, cloud computing faces certain challenges and issues that hinder widespread adoption of cloud. For instance, security, performance, and quality are a few to mention [7.10, 7.27].

The development and operation of data storage sites is ongoing process in organizations. Off-site data storage is a cloud application that liberates the customer from focusing on data storage systems [7.10]. Representing system characteristics and capabilities as utility, causes the user to focus on aspects directly related to data (security, transmission, processing) [7.6, 7.33]. However, moving data to the cloud, administered and operated by certain vendor requires high level of trust and security. Multiple users, separated through logical barriers of virtual machines, share resources including storage space. Multi-tenancy and virtualization generate risks and underpins the confidence of users to adopt the cloud model [7.2, 7.3]. Armbrust *et al.* [7.1] ranked data confidentiality and auditing at number three in the list of top ten obstacles impeding widespread cloud adoption. Data can be used by the cloud service providers without authorization [7.2, 7.23, 7.4] and can be accessed by other machines in cloud [7.23, 7.3].

Data being the principal asset for organizations needs to be secured. Especially, when data must enter a public cloud. To avoid unauthorized access to the cloud data, access control mechanism must be enforced [7.16, 7.17]. Moreover, data leakage and data privacy strategies must be employed so that only authorized users can access and utilize data. Refraining cloud service providers from utilizing the customer data requires high preventive measures [7.3]. Encryption techniques provide a solution to ensure privacy and confidentiality of stored data. However, key management becomes a prime issue in the case of encryption [7.28, 7.31]. Cryptographic keys need to be stored and protected. Compromise or failure of a key storage facility may lead to the loss of data. Therefore, cryptographic keys must be stored in a robust manner and a single point of failure should not affect the availability of data [7.31].

The security concerns of outsourcing data to public clouds, serves as our motivation to work for the development of data security technique. We aim for a technique capable of addressing the aforementioned critical issues. We propose a data security scheme that uses key manager servers for the management of cryptographic keys. Shamir's (*k*, *n*) threshold scheme [7.26] is used for the management of keys that uses *k* shares out of *n* to rebuild the key. Access to key and data is ensured through a policy file that states policies under which access is granted to the keys. The client generates random symmetric keys for encryption and integrity functions. Symmetric keys are protected by the public key generated by the key manager(s) (Fig. 7.1). All of the symmetric keys are deleted from the client afterwards. Encrypted data and keys are uploaded to the cloud. For downloading the data, client presents a policy file to cloud and

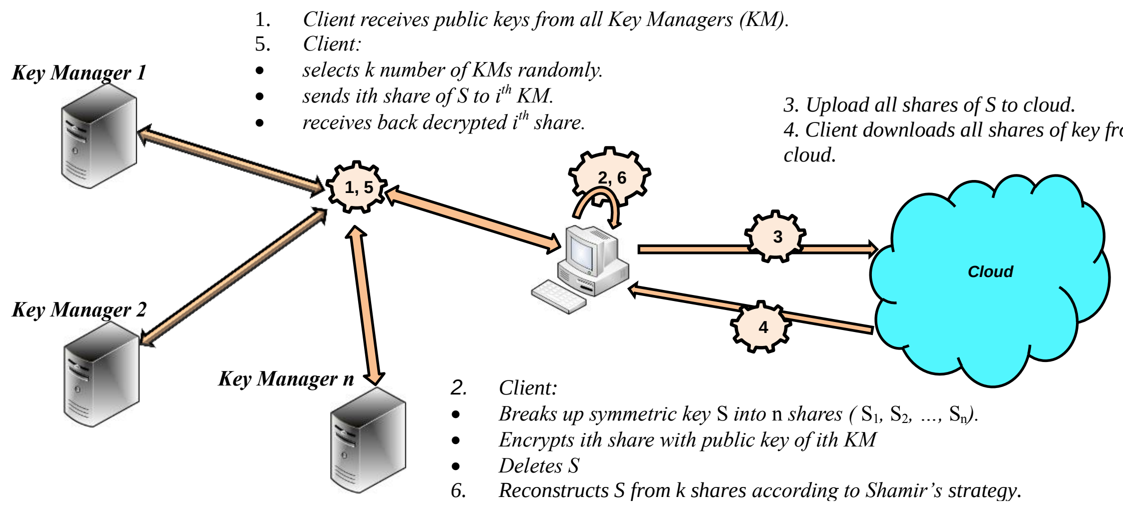

1. Client receives public keys from all Key Managers (KM).
5. Client:
• selects k number of KMs randomly.
• sends ith share of S to i^{th} KM.
• receives back decrypted i^{th} share.
Key Manager 1
3. Upload all shares of S to cloud.
4. Client downloads all shares of key from cloud.
1, 5
2, 6
3
Cloud
4
Key Manager 2
Key Manager n
2. Client:
• Breaks up symmetric key S into n shares ($S_1, S_2, \ldots, S_n$).
• Encrypts ith share with public key of ith KM
• Deletes S
6. Reconstructs S from k shares according to Shamir's strategy.

Fig. 7.1. Shamir's (*k, n*) Threshold Scheme in DaSCE.

downloads the encrypted data and keys. Keys are decrypted by key manager(s). Thereafter, the client decrypts the data.

We review the scheme presented in [7.29], called File Assured Deletion (FADE). The FADE is a light-weight and scalable technique that assures deletion of files from cloud when user asks for deletion. However, during our analysis, FADE fell short on issues of security of keys and authentication of participating parties. Based on our analysis and issues identified with FADE, we propose enhancements to the scheme and name it as Data Security for Cloud Environment with Semi-Trusted Third Party (DaSCE) that enhances the security of keys and authentication process. Moreover, to mitigate the man-in-the-middle-attack, we included supplementary steps for the session key establishment process. The aforesaid steps augment the security level and prohibit the malicious user to carry out the attack at slight performance overheads. However, the results from our verification analysis revealed that DaSCE is more secure than FADE when man-in-the-middle attack was introduced. Our major contributions include:

- Development of a security scheme (DaSCE) for outsourced data to cloud that uses a combination of symmetric and asymmetric encryption. The DaSCE ensures data confidentiality at a cloud infrastructure, as long as it is in use by the client. It also assures that data gets deleted and becomes unrecoverable after the user deletes it from the cloud.
- Enforcing access control to both data and key through validity of policies and mutual authentication between client and key managers, and client and cloud. Digital signatures and variation of Diffie-Hellman is used for mutual authentication of parties. Successful authentication and session key establishment results in access to asymmetric keys that are used in subsequent cryptographic operations.
- Ensuring the integrity of data by use of symmetric key and message authentication code and securing symmetric keys with asymmetric keys generated by third party key managers.
- Formal modeling and verification of FADE and DaSCE by using High Level Petri Nets (HLPN), SMT-Lib, Z3 solver, and Scyther.
- We implemented a prototype of DaSCE and evaluated the performance of DaSCE based on time consumption parameters (file upload time, file download time, cryptographic operations time).

7.2. File Assured Deletion (FADE)

The FADE protocol provides privacy, integrity, access control, and assured deletion to outscored data. The FADE uses both symmetric and asymmetric keys. Symmetric keys are protected by using Shamir's (*k*, *n*) scheme to ample the trust level in the key. The FADE works with a group of key managers (KM). Following keys are used by FADE protocol. The variable *K* is termed as data key and is used to encrypt file *F* of the client and *S* as secret key that is used to encrypt *K*. The public/private key pair generated by *KMs* is represented by (e_i, d_i) and is used to encrypt *S*. The *K* and *S* are symmetric keys. The operations supported by FADE are: **(a)** File upload, **(b)** File download, **(c)** Policy Revocation, and **(d)** Policy Renewal. The aforementioned operations are explained below. The notations used in the paper are presented in Table 7.1.

Table 7.1. Notations and Their Meanings.

Notation	**Meanings**
KM	Key manager
F	File
K	A symmetric key
S	A symmetric key.
e_i	Public key parameter.
n_i	Public key parameter.
d_i	Private key parameter.
e_j	Modified/New public key parameter.
n_j	Modified/New public key parameter.
d_j	Modified/New private key parameter.
$\{F\}_K$	File encrypted with key *K*.
$\{K\}_S$	*K* encrypted with key *S*.
S^e	*S* encrypted with public key *e*.
MAC	Message Authentication code
HMAC	Hash-based MAC
P_i	Original policy file of client
P_j	Modified policy file
HLPN	High Level Petri Net
IK	Integrity key for MAC calculation

7.2.1. File Upload

When data must be uploaded to the cloud, the client requests the *KM* to generate a public/private key pair. The said is done by sending a policy file, P_i, to the *KM*. The *KM* generates the key pair, associates that with the P_i, and sends the public part of the key (e_i, n_i) to the client. After receiving public key for P_i, the client performs the following cryptographic operations. The client encrypts F with K to generate $\{F\}_K$ (F encrypted with K). The K is then encrypted with S_i to get $\{K\}_{Si}$. Subsequently, S_i is encrypted with the public key generated by the *KM* with $P_{i.}$ The S_i is encrypted using asymmetric encryption (S_i^{ei} *mod n*). The P_i, $\{F\}_K$, $\{K\}_{Si}$, and (S_i^{e} *mod n*) are uploaded to the cloud afterwards. The hashed MAC (HMAC) of data file is also uploaded with the encrypted file. The client deletes all of the symmetric keys through secure overwriting. The process of file upload is shown in the Figure 2(a).

When FADE works with full quorum of *KMs*, S_i is divided into n shares and each share is encrypted with a public key generated by one of the *KMs*. The key is divided based on Shamir's

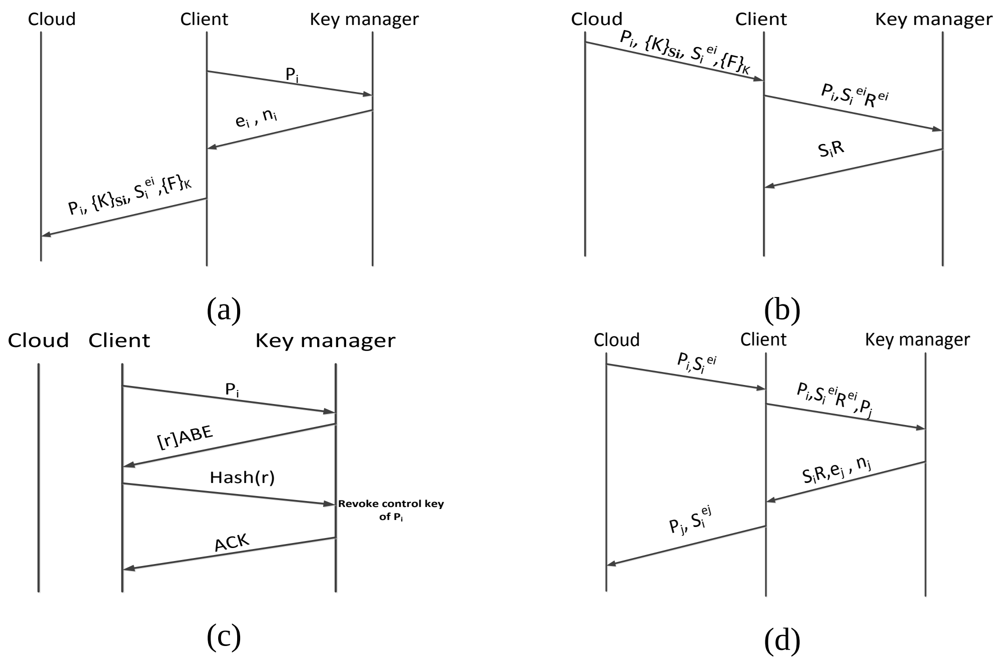


Fig. 7.2. FADE (a) File Upload, (b) File Download, (c) Policy Revocation, and (d) Policy Renewal (single key manager) [7.29].

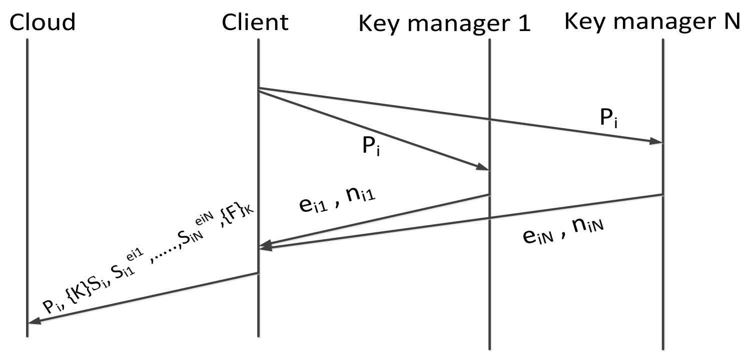


Fig. 7.3. Fade File Upload with Multiple Key Managers [7.29].

(*k*, *n*) threshold scheme. To get back the S_i, *k* shares are needed. The FADE protocol does not authenticate the client for the upload process. The process with multiple *KMs* is shown in Fig. 7.3. When data must be uploaded to the cloud, the client requests the *KM* to generate a public/private key pair. The said is done by sending a policy file, P_i, to the *KM*. The *KM* generates the key pair, associates that with the P_i, and sends the public part of the key (e_i, n_i) to the client. After receiving public key for P_i, the client performs the following cryptographic operations. The client encrypts *F* with *K* to generate $\{F\}_K$ (*F* encrypted with *K*). The *K* is then encrypted with S_i to get $\{K\}_{Si}$. Subsequently, S_i is encrypted with the public key generated by the *KM* with P_i. The S_i is encrypted using asymmetric encryption (S_i^{ei} *mod n*). The P_i, $\{F\}_K$, $\{K\}_{Si}$, and (S_i^e *mod n*) are uploaded to the cloud afterwards. The hashed MAC (HMAC) of data file is also uploaded with the encrypted file. The client deletes all of the symmetric keys through secure overwriting. The process of file upload is shown in the Figure 2(a).

When FADE works with full quorum of *KMs*, S_i is divided into *n* shares and each share is encrypted with a public key generated by one of the *KMs*. The key is divided based on Shamir's (*k*, *n*) threshold scheme. To get back the S_i, *k* shares are needed. The FADE protocol does not authenticate the client for the upload process. The process with multiple *KMs* is shown in Fig. 7.3.

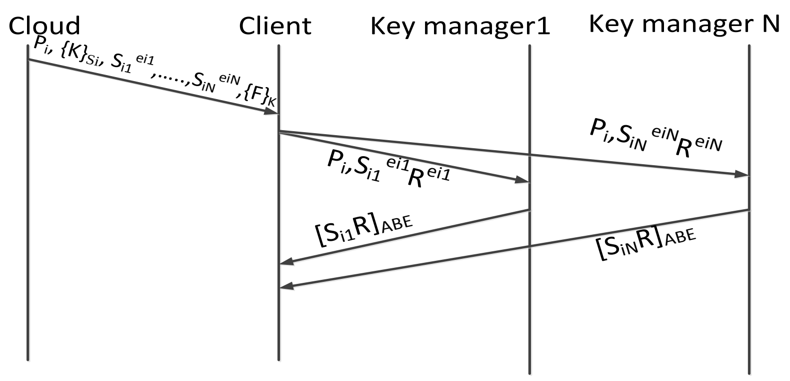

Cloud
Client
Key manager1
Key manager N
P_i, $\{K\}_{Si}$, $S_{i1}{}^{ei1}$,.....,$S_{iN}{}^{eiN}$,$\{F\}_K$
P_i,$S_{iN}{}^{eiN}R^{eiN}$
P_i,$S_{i1}{}^{ei1}R^{ei1}$
$[S_{i1}R]_{ABE}$
$[S_{iN}R]_{ABE}$

Fig. 7.4. File Download Using ABE with Multiple Key Managers [7.29].

7.2.2. File Download

The client requests the cloud for file and encrypted keys to download. The client checks for the integrity of the file through the HMAC. Afterwards, the client generates a secret number *R* and calculates R^{ei} and then generates $S_i^e R^{ei} = (S_iR)^{ei}$. The $(S_iR)^{ei}$ is then sent to KM for decryption. The *KM* decrypts $(S_iR)^{ei}$ with corresponding d_i and sends back S_iR. At this point, ABE comes into the play. The *KM* sends S_iR with ABE, where the attributes used for ABE are based on P_i. The client extracts S_i from the received message and decrypts *K* that is used to decrypt *F*. The process is highlighted in Fig. 7.2(b). Similarly, the file download with multiple *KMs* takes place according to the flow of messages shown in Fig. 7.4.

7.2.3. Policy Revocation

If P_i needs to be revoked, the client requests the *KM* by sending the P_i. The *KM* generates a random number *r* and sends *r* to the client after encryption with ABE. The authentic client decrypts *r*, calculates the hash value, and sends back to the *KM*. After verification the *KM* revokes *Pi* and acknowledges the client as depicted in Fig. 7.2 (c).

7.2.4. Policy Renewal

If P_i needs to be renewed as P_j, the client downloads all of the keys and sends P_i and encrypted S_i to the *KM* along with P_j. The *KM* decrypts S_i. Moreover, the *KM* sends new public key parameters (e_j, n_j) to the client as outlined in Fig. 7.2 (d). We will now formally analyze FADE in the following section.

7.3. Analysis of FADE

The FADE is a light weight protocol that does not require heavy modifications in cloud architecture. The analysis of FADE identified the following issues.

7.3.1. File Upload

In file upload process of FADE we assume that there is a man-in-the-middle (intruder) between client and *KM*. The intruder can intercept P_i and send P_j (modified P_i) to *KM*. In the second step, the *KM* sends (e_i, n_i). The intruder intercepts (e_i, n_i) and sends the client modified parameters (e_j, n_j). The client encrypts the keys with (e_j, n_j) and uploads to the cloud. The client cannot verify that the received (e_j, n_j) is from *KM* or any other entity. The aforesaid issue is highlighted in Fig. 7.5 (a).

In the original file upload process of FADE, independence of Step 1 and Step 2 allows the intruder to carry out the attack. The policies received by the *KM* are neither from the client nor does the client receive keys from the *KM*. However, both assume a valid data exchange with each other. As a result, the client encrypts the S_i with the (e_j, n_j). The encryption of data with the intruder's generated keys may result in any of the following scenarios:

7.3.1.1. Intruder Fetches the Data during Download Process

During the download process, the intruder can intercept the data. As S_i is encrypted with (e_j, n_j) that is generated by the intruder; therefore, after reviving S_i^{ej} intruder can recover S_i by decryption with a corresponding d_j as: $S_i = (S_i^{ej})d_j \ mod \ n$. Once S_i is decrypted, the intruder can easily decrypt *K* and gather *F*.

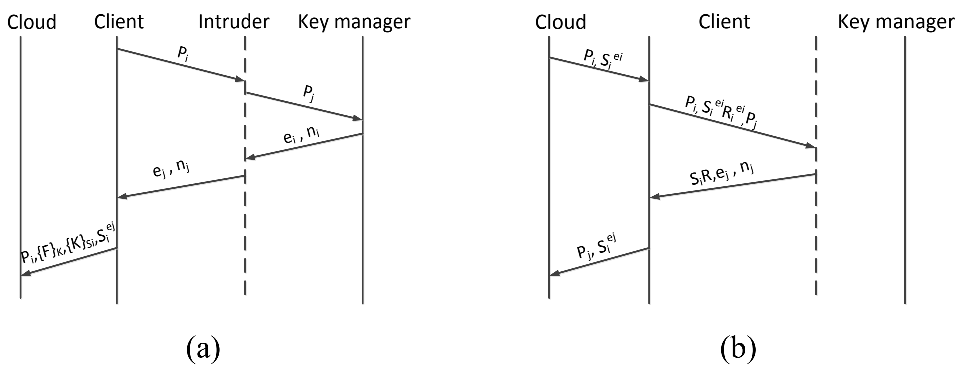


Fig. 7.5. (a) Man-in-the-middle Attack that Causes Encryption with the Wrong Keys (b) Exploitation of Policy Renewal Process.

7.3.1.2. Intruder Stays Aside during Download Operation

The client downloads the data from the cloud and sends S_i to the *KM* for decryption. As S_i was encrypted by the public key that was originally generated by the intruder, the *KM* will not be able to decrypt the correct S_i. Therefore, access to the data will be denied. The denial of access will result in the loss of data. The *KM* generates the keys based on ABE having policies defined in P_i. During the attack, the *KM* generates the keys with P_j (modified P_i). Therefore, even the attributes will not correspond to the original policies. Same attack flow can be modeled for multiple *KMs* as shown in Fig. 7.6. As highlighted in Fig. 7.6, all S_i*'s* are encrypted with keys generated by the intruder and the corresponding d_i*'s* are held by the intruder. Therefore, the intruder can generate S_i. However, intruder must intercept *k* portions of S_i.

7.3.2. Policy Renewal

Fig. 7.5 (b) shows how the intruder can exploit the policy renewal process of FADE for denying access of data to a legitimate user. It is noteworthy to mention that the exploitation is only possible if initially the attack depicted in Fig. 7.5 (a) is already carried out. The client after downloading S_i from the cloud sends S_i along with P_i to the *KM*. The intruder intercepts the data, decrypts S_i with the corresponding private key, generates a new pair of public/private key, and

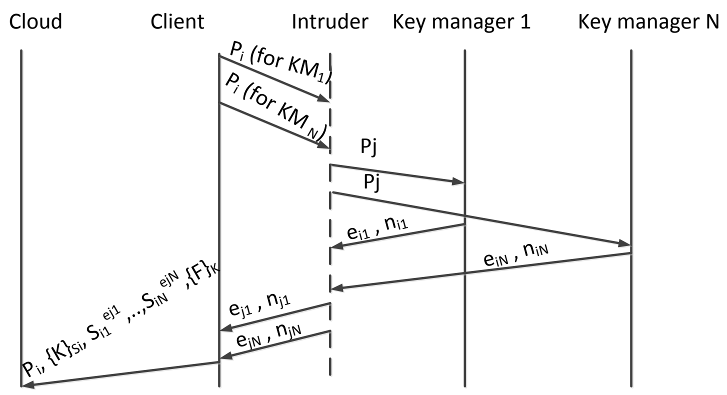


Fig. 7.6. Man-in-the-middle with Multiple Key Managers.

sends it to the client. The client performs cryptographic operations (as it did earlier) and uploads the data and keys to the cloud.

7.3.3. Attack Verification through Scyther

In this section, we verify the attack defined in the previous section using Scyther, which is a graphical tool for analysis, verification, and falsification of security protocols [7.5]. We

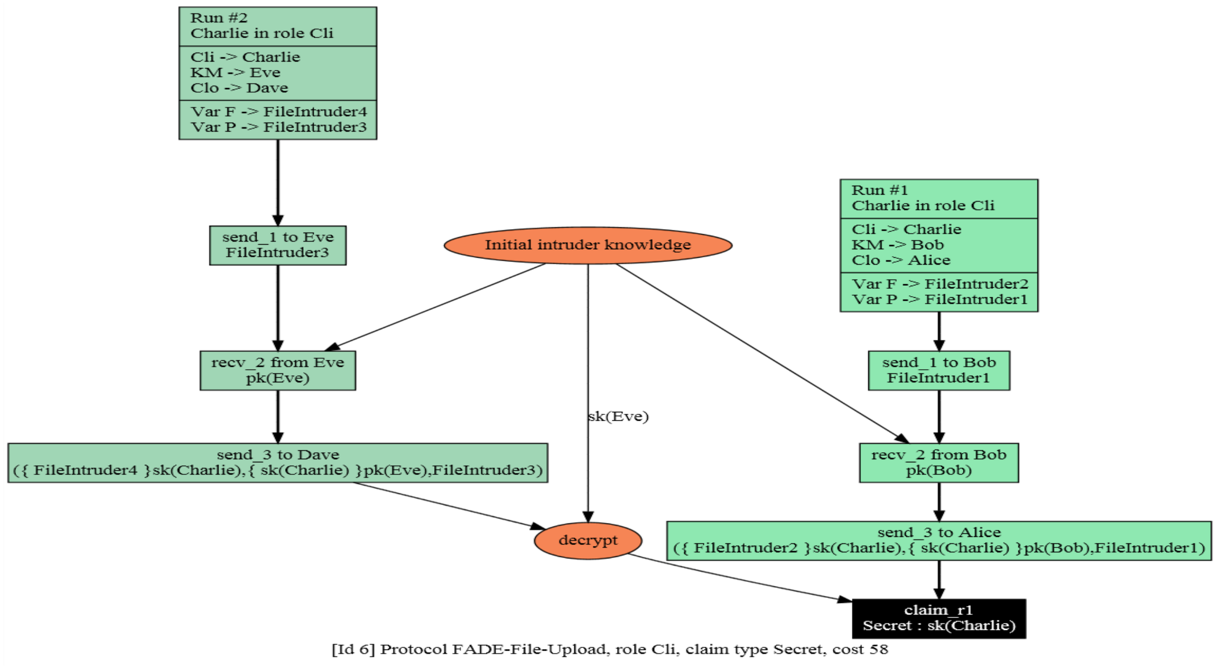


Fig. 7.7. Scyther Verification of FADE.

modeled FADE in Scyther and verified whether S_i and F remain secret under the setup or otherwise. The verification is performed by a "claim" (see Fig. 7.7) that S_i remains secret during the process. The Scyther verified the validity of the claim and reported the attack that was discussed in Section 4.1.

In Scyther, Charlie plays a role of a client, Bob as the *KM*, Alice as the cloud, and Eve as the intruder. The Run# 1 of the Scyther is not an intercepted run while Run#2 is a run where intruder plays the part. Eve intercepts the P_i and sends Charlie the generated public key *pk*(*Eve*). Later on Eve can use corresponding private key *sk*(*Eve*) to decrypt the secret key of Charlie (*sk*(*Charlie*)). In this model, *sk*(*Charlie*) is the same key as S_i, in the explained model. Our claim that S_i will remain secret is falsified by Scyther by producing the counter attack.

7.3.4. HLPN

Petri Nets provide graphical and mathematical representation of the system and can be applied to variety of systems for instance stochastic, deterministic, and asynchronous computations [7.24]. A HLPN is a 7-tuple $N = (P, T, F, \varphi, R_n, L, M_0)$, where P is set of places; T refers to the set of transitions such that $P \cap T = \emptyset$; Flow relations are defined by F such that $F \subseteq (P \times T) \cup (T \cup P)$; φ maps places P to the data types. Rules for transitions are defined by R_n; L is a label on F and M_0 represents the initial marking [7.24]. In the above definition, the structure of the Petri Net is given by P, T, and F; whereas, (φ, R_n, L) provide the static semantics of the Petri Net model.

7.3.5. SMT-Lib and Z3 Solver

SMT has roots in Boolean Satisfiability Solvers (SAT) [7.11, 7.12, and 7.22]. SMT-Lib provides a common input platform and benchmarking framework that helps in the evaluation of the systems. We use Z3 solver with SMT-Lib that is a theorem prover developed at Microsoft

Research. Z3 is an automated satisfiability checker. In addition, Z3 determines whether the set of formulae are satisfiable in the built-in theorems of SMT-Lib [7.21].

7.3.6. Verification through HLPN Model

In this section, we formally analyze the man-in-the-middle attack on FADE protocol. We use High Level Petri Nets (HLPN) and Z language [7.8, 7.11, 7.12, 7.22, 7.24, and 7.25] to perform formal analysis. HLPN define mathematical properties for the system and simulate the system to analyze the behavior. We verify HLPN model of FADE using Satisfiability Modulo Theories Library (SMT-Lib) and Z3 solver. To verify the model, the Petri Net model is first translated into SMT along with the specified properties. Subsequently, Z3 solver is used to determine whether or not the properties hold.

7.3.7. Formal Verification

The verification process checks for the correctness of the system. In model checking: (a) description of the system is provided stating properties or rules of the system, (b) system is represented by a model, and (c) some verification tool is used to check whether the model holds the specified properties or not. In this paper we use the bounded model checking to verify the man-in-the-middle attack on FADE.

The HLPN model for FADE is given in Fig. 7.8. The model is given with the intruder between the client and *KM*. The data types used in the model and their mappings are shown in Table 7.2 and Table 7.3, respectively. All the rectangular black boxes in HLPN are transitions and belong to the set *T*. The circles are places and belong to the set *P*.

Table 7.2. Data Types Used in FADE HLPN Model.

Types	**Description**
Policy	A string type for describing file access policy.
File	A string type holding data to be protected.
K	A string type representing symmetric key.
S	A string type representing symmetric key.
e	Public Key parameter.
n	Public Key parameter.
d	Private Key parameter.
$\{F\}_K$	File encrypted with key K.
$\{K\}_S$	K encrypted with key S.
S^e	S encrypted with public key e.

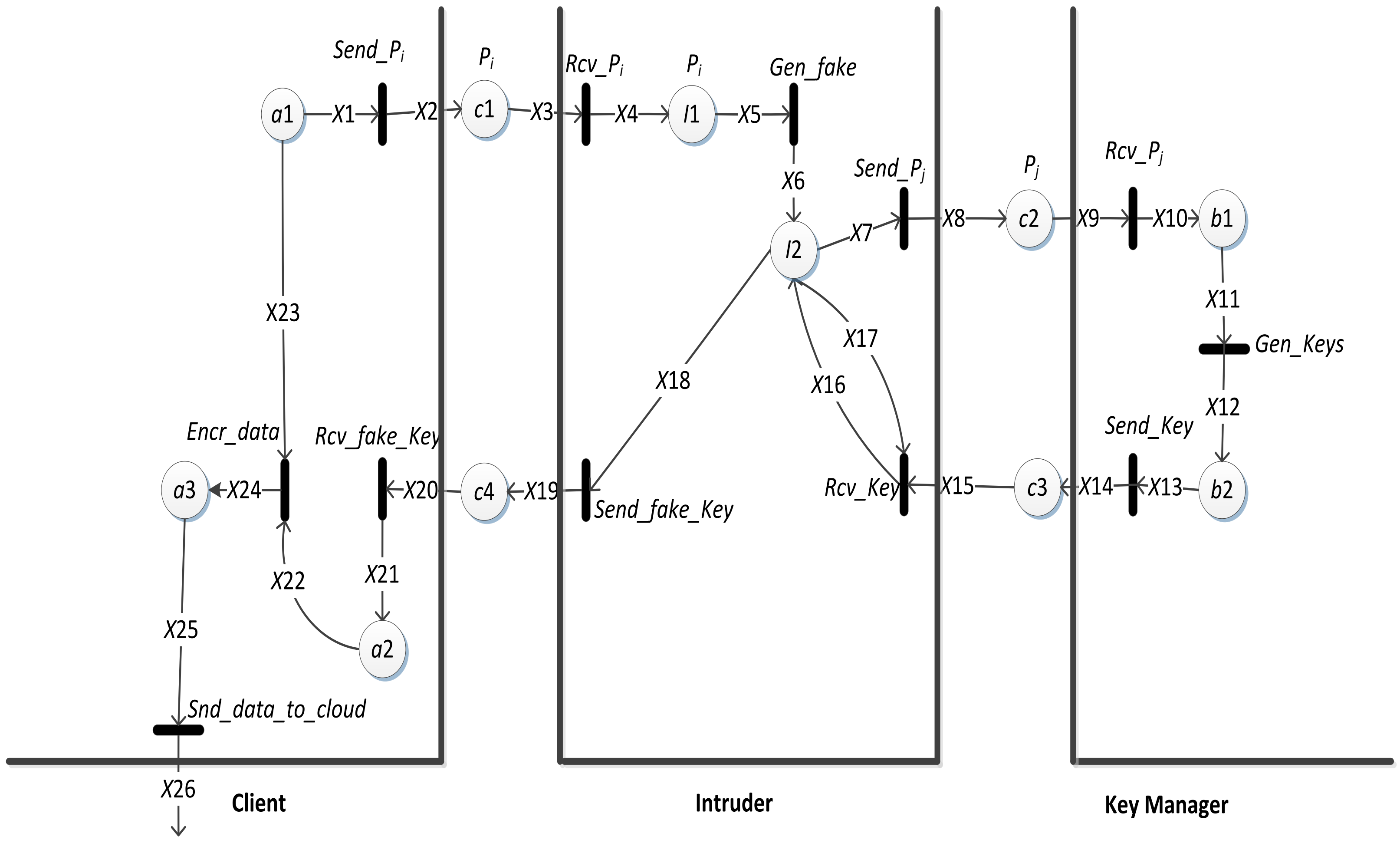


Fig. 7.8. FADE HLPN Model with Intruder.

The process starts with the client sending *P_i* to the *KM*. The file is intercepted by the intruder. The file sending and receiving is performed on transitions *Send_P_i* and *Rcv_P_i*. Rule (7.1) and Rule (7.2) are mapped to the aforesaid transitions.

$$R(Send\_P_i) = \forall\, x_1 \in X_1, \forall\, x_2 \in X_2 |\, x_2 \coloneqq x_1[1] \wedge \\ X_2' = X_2 \cup \{x_2\}, \tag{7.1}$$

$$R\left(Rcv_{P_i}\right) = \forall\, x_3 \in X_3, \forall\, x_4 \in X_4 |\, x_4 \coloneqq x_3 \wedge \\ X_4' = X_4 \cup \{x_4\}. \tag{7.2}$$

The intruder generates P_j and sends it to the *KM*. The transition *Gen_fake* is fired upon interception of original P_i. Following are the three transition and the corresponding rules.

$$\boldsymbol{R}\left(Gen_{fake}\right) = \forall\, x_5 \in X_5, \forall\, x_6[2] \in X_6, \\ \forall\, x_6[3] \in X_6, \forall\, x_6[4] \in X_6, \\ \forall\, x_6[5] \in X_6 |\, x_5 \leftrightarrow X_6[2] \leftrightarrow X_6[3] \leftrightarrow X_6[4] \leftrightarrow X_6[5] \wedge \\ X_6' = X_6 \cup \{x_5, x_6[2], x_6[3], x_6[4], x_6[5]\}, \tag{7.3}$$

$$\boldsymbol{R}\left(Send_{P_j}\right) = \forall\, x_7[2] \in X_7, \forall\, x_8 \in X_8 | \\ x_8 \coloneqq X_7[2] \wedge \\ X_8' = X_8 \cup \{x_8\}, \tag{7.4}$$

$$\boldsymbol{R}\left(Rcv_{P_j}\right) = \forall\, x_9 \in X_9, \forall\, x_{10} \in X_{10} |\, x_{10} \coloneqq X_9 \wedge \\ X_{10}' = X_{10} \cup \{x_{10}\}. \tag{7.5}$$

The keys generated and sent by KM are intercepted by the intruder. The Following rules (7.6) and (7.7) capture the above three transitions.

Table 7.3. Mapping of Data Types and Places.

Types	**Description**
φ(a1)	$\mathbb{P}$(Policy × File × K × S)
φ(c1)	$\mathbb{P}$(Policy)
φ(I1)	$\mathbb{P}$(Policy)
φ(I2)	$\mathbb{P}$(Policy ×Policy × e × n × d × e × n)
φ(c2)	$\mathbb{P}$(Policy)
φ(b1)	$\mathbb{P}$(Policy)
φ(b2)	$\mathbb{P}$(Policy × e × n × d)
φ(c3)	$\mathbb{P}$(e × n)
φ(c4)	$\mathbb{P}$(e × n)
φ(a2)	$\mathbb{P}$(e × n)
φ(a3)	$\mathbb{P}$(Policy ×$\{F\}_K$ × $\{K\}_S$ × Se)

$$\boldsymbol{R}(Gen\_Keys) = \forall x_{11} \in X_{11}, \forall x_{12}[1] \in X_{12}, \forall x_{12}[2] \in X_{12}, \forall x_{12}[3] \in X_{12},$$

$$\forall x_{12}[4] \in X_{12} | \forall x_{12}[1] \coloneqq X_{11} \wedge x_{12}[1] \leftrightarrow X_{12}[2] \leftrightarrow X_{12}[3] \leftrightarrow X_{12}[4] \wedge \qquad (7.6)$$

$$X'_{12} = X_{12} \cup \{x_{12}[1], x_{12}[2], x_{12}[3], x_{12}[4]\},$$

$$\boldsymbol{R}(Send\_Key) = \forall x_{13}[2] \in X_{13}, \forall x_{13}[3] \in X_{13}, \forall x_{14} \in X_{14} | \qquad (7.7)$$

$$x_{14}[1] \coloneqq x_{13}[2] \wedge x_{14}[2] \coloneqq x_{13}[3] \wedge$$

$$X'_{14} = X_{14} \cup \{x_{13}[2], x_{13}[3]\},$$

$$\boldsymbol{R}(Rcv\_Key) = \forall x_{15} \in X_{15}, \forall x_{17}[1] \in X_{17}, \forall x_{17}[2] \in X_{17}, \forall x_{17}[3] \in \underset{ss}{X_{17,}} \qquad (7.8)$$

$$\forall x_{17}[4] \in \underset{ss}{X_{17,}} \forall x_{17}[5] \in \underset{ss}{X_{17,}} \forall x_{17}[6] \in \underset{ss}{X_{17,}} \forall x_{17}[7] \in X_{17} |$$

$$x_{17}[2] \leftrightarrow X_{15}[1] \leftrightarrow x_{15}[2] \wedge$$

$$X'_{16} = X_{16} \cup \{x_{17}[1], x_{17}[2], x_{17}[3], x_{17}[4], x_{17}[5], x_{15}[1], x_{15}[2]\}.$$

The intruder generates and sends (e_j, n_j) to the client as depicted in (7.9) and (7.10).

$$\boldsymbol{R}(Send\_fake\_Key) = \forall x_{18}[3] \in X_{18}, \forall x_{18}[4] \in X_{18}, \forall x_{19} \in X_{19} |$$

$$x_{19}[1] \coloneqq X_{18}[3] \wedge x_{19}[2] \coloneqq X_{18}[4] \wedge X'_{19} = X_{19} \cup \{x_{18}[3], x_{18}[4]\}, \qquad (9)$$

$$(Rcv\_fake\_Key) = \forall x_{20} \in X_{20}, \forall x_{21}[4] \in X_{21} | \quad (7.10)$$
$$x_{21}[1] \coloneqq X_{20}[1] \wedge x_{21}[2] \coloneqq x_{20}[2] \wedge$$
$$X'_{21} = X_{21} \cup \{x_{21}[1], x_{21}[2]\},$$

The client performs the cryptographic operations with (e_j, n_j) and sends all the encrypted data to the cloud. This is represented by the following rules.

$$\boldsymbol{R}(Encr\_data) = \forall x_{21} \in X_{21}, \forall x_{22} \in X_{22}, \forall x_{23} \in X_{23}, \forall x_{24} \in X_{24} |$$
$$x_{24}[1] \coloneqq \underset{ss}{x_{23}}[1] \wedge x_{24}[2] \coloneqq x_{23}[2] encr(x_{23}[3]) \wedge x_{24}[3] \quad (7.11)$$
$$\coloneqq x_{23}[3] encr(x_{23}[4] \wedge$$
$$x_{24}[4] \coloneqq x_{23}[4] encr(x_{21}[1], x_{22}[2]) \wedge$$
$$X'_{24} = X_{24} \cup \{x_{24}[1], x_{24}[2], x_{24}[3], x_{24}[4]\},$$
$$\boldsymbol{R}(Snd\_data\_to\_Cloud) = \forall x_{25} \in X_{25}, \forall x_{26} \in X_{26} | x_{26}[1] \coloneqq x_{25}[1] \wedge$$
$$x_{26}[2] \coloneqq x_{25}[2] \wedge x_{26}[3] \coloneqq x_{25}[3] \wedge x_{26}[4] \coloneqq x_{25}[4] \wedge \quad (7.12)$$
$$X'_{26} = X_{26} \cup \{x_{26}[1], x_{26}[2], x_{26}[3], x_{26}[4]\}.$$

In the above, *Encr_data* is the most crucial transition. Security of data and the keys are highly dependent on this transition. If the encryption is performed by using (e_j, n_j), then the data security is compromised. In this context, the property that we verified using SMT-Lib and Z3 is that: if the intruder is present, then the encryption operation is performed using the wrong keys. The property of the model is described using a formal language called Computational Tree Logic (CTL*). The CTL* uses numerous temporal operators to represent various operations [7.7, 7.20]. For instance, *A* represents "for all paths", *G* denotes "globally", and *F* characterizes "future state". The property specified in CTL* using temporal operators is given as: $AG(a1 \rightarrow AF\ a3)$. After translating the above model into SMT-Lib, we performed bounded checking using Z3 solver. The mentioned property was satisfied by the solver in 310 msec.

7.4. DaSCE

From Section 4, it is evident that the security of S_i in FADE depends on the key exchange between the client and the *KM*. If the key exchange is compromised, then S_i is compromised, that in turn, leaks all the keys and the data. We observed that the reason for the said attack is the independence of communication steps between the client and the *KM* that allows the attacker to launch the attack and subvert the whole process. In this section, we propose improvements in the communication process between **(a)** client and the *KM*, and **(b)** client and the cloud. Our proposed changes link the communication steps so as to avoid attacker to overtake the process.
We use the station-to-station (STS) protocol [7.9] and digital signature for authentication and session key establishment before any other exchange takes place. The keys generated by the *KMs* and policy files are exchanged using session keys. Some modifications are required in the subsequent operations of the protocol as the session keys are introduced to the FADE. The following subsections discuss the proposed mechanisms.

7.4.1. DaSCE Keys

The DaSCE makes use of both symmetric and asymmetric keys. The confidentiality and integrity services for data are provided through symmetric keys that are secured by using asymmetric keys. Asymmetric key pairs are generated by third party KMs. Out of the key pair, only public key is transmitted to the client. For secure transmission of keys, a session key is established between client and KM through STS protocol. To avoid man-in-the-middle attack, both client and KM are authenticated by use of digital signatures. As a new session key is used for every communication session between client and KM, the session key is exchanged through

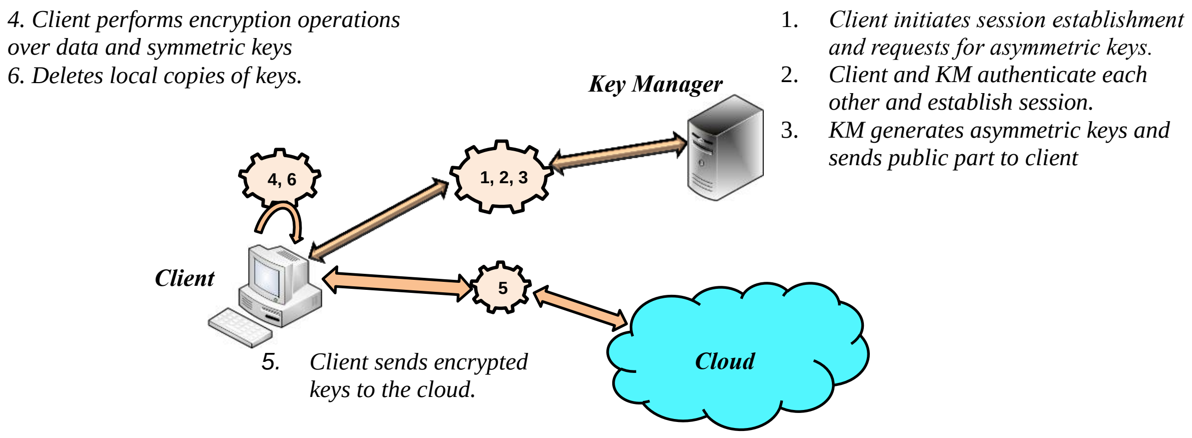

4. Client performs encryption operations over data and symmetric keys
6. Deletes local copies of keys.
Key Manager
1. Client initiates session establishment and requests for asymmetric keys.
2. Client and KM authenticate each other and establish session.
3. KM generates asymmetric keys and sends public part to client
4, 6
1, 2, 3
Client
5
5. Client sends encrypted keys to the cloud.
Cloud

Fig. 7.9. Key Management in DaSCE

key exchange process and is not randomly generated. This also avoids weakness of randomly generated keys. The symmetric keys are generated once for data encryption by client and encrypted by another symmetric key named S_i. The S_i is finally protected by the public key received from KM. The encrypted keys are stored at cloud and client deletes the local copies of the keys. For decryption purpose, client establishes a session with KM and sends S_i to KM after masking with random number R. The KM decrypts S_i and sends back to client. The client unmasks S_i to get the symmetric keys. Fig. 7.9 depicts the key management process.

7.4.2. File Upload

For the establishment of session key, we assume that the parameters required are fixed and publically available to all of the users. We call these parameters as α and p where, α is a large number known as the primitive root and p is a large prime number. The process comprises of following steps.

- The client generates a random number x and calculates $\alpha^x \bmod p$ and sends to the *KM*.
- The *KM* generates a random number y and calculates $\alpha^y \bmod p$. The *KM* also calculates $(\alpha^x)^y$ as a session key, *EK*, between client and *KM*.

- The *KM* generates digital signature over {α^y, α^x} ($S_{KM}\{\alpha^y, \alpha^x\}$) and encrypts it with the generated session key to generate $EK(S_{KM}\{\alpha^y, \alpha^x\})$.
- The *KM* sends (α^y , $EK(S_{KM}\{\alpha^y, \alpha^x\})$) to the client.
- The client verifies the signature using the public key of the *KM* and calculates the session key as $(\alpha^y)^x$.
- The client calculates $EK(S_{Cli}\{ \alpha^x , \alpha^y \})$ and encrypts P_i with *EK* and sends both of the calculated values to the *KM*. The sent message contains $EK(S_{Cli}\{ \alpha^x , \alpha^y \})$, $EK(P_i)$.
- The *KM* verifies the signature of the client. Upon successful verification, the *KM* decrypts P_i and generates (e_i, n_i) with P_i. The *KM* stores the decrypted P_i.
- The *KM* encrypts (e_i, n_i) with the *EK* to generate ($EK(e_i, n_i)$), *which is sent to the client.*
- *The client encrypts the file F* with key *K*, calculates MAC with *IK*; and encrypts *K* and *IK* with S_i. Afterwards S_i is encrypted with e_i. Subsequently, the client sends all the encrypted data to cloud.
- The client erases all of the keys except public key parameters received from the *KM*.

The file upload process is shown in Fig. 7.10. The calculations for session key include *mod p* operation which is not shown in the figure for clarity.

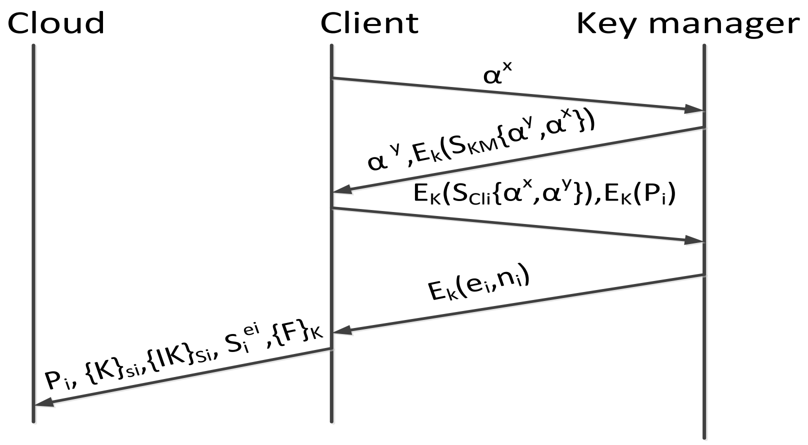

Cloud
Client
Key manager
α^x
$\alpha^y, E_K(S_{KM}\{\alpha^y, \alpha^x\})$
$E_K(S_{Cli}\{\alpha^x, \alpha^y\}), E_K(P_i)$
$E_K(e_i, n_i)$
$P_i, \{K\}_{Si}, \{IK\}_{Si}, S_i^{ei}, \{F\}_K$

Fig. 7.10. DaSCE File Upload with Single Key Manager.

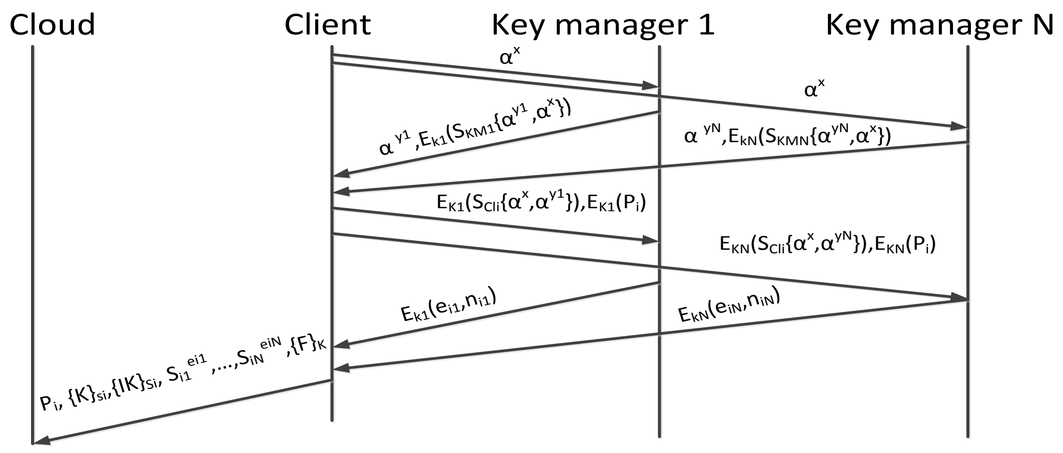

Cloud
Client
Key manager 1
Key manager N
α^x
α^x
α^y1,E_k1(S_KM1{α^y1,α^x})
α^yN,E_kN(S_KMN{α^yN,α^x})
E_K1(S_Cli{α^x,α^y1}),E_K1(P_i)
E_KN(S_Cli{α^x,α^yN}),E_KN(P_i)
E_k1(e_i1,n_i1)
E_kN(e_iN,n_iN)
P_i, {K}_si,{IK}_Si, S_i1^ei1,...,S_iN^eiN,{F}_K

Fig. 7.11. DaSCE File Upload with Multiple Key Managers.

Similarly, the file upload process with multiple *KMs* is shown in Fig. 7.11. With multiple *KMs*, S_i is divided into n shares and each share is encrypted with the key from one of the managers according to (k, n)-threshold scheme. The interdependencies between file upload steps circumvent the man-in-the-middle attack. If higher level of security is required, then session key can also be established between the client and the cloud to keep the P_i exchange secure.

7.4.3. File Download

The file download process of DaSCE is depicted in Fig. 7.12. The process starts with the

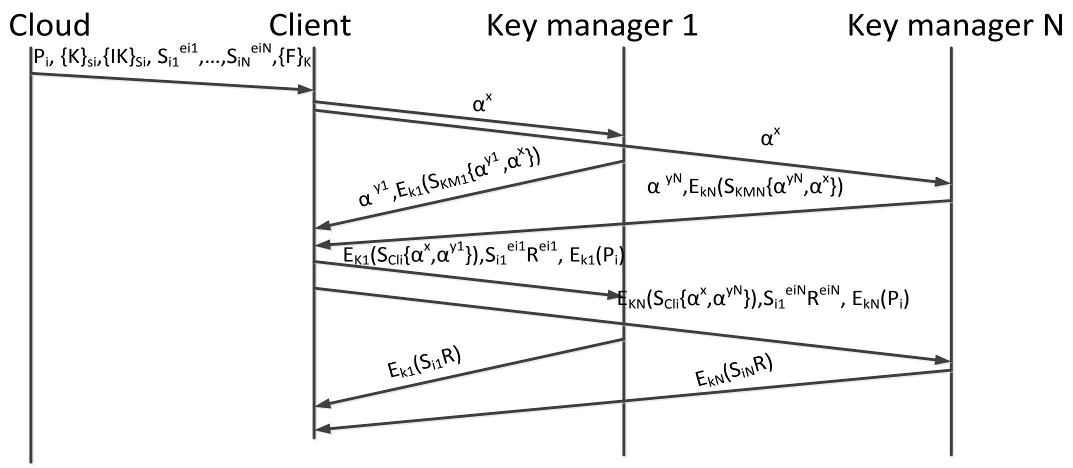

Cloud
Client
Key manager 1
Key manager N
P_i, {K}_si,{IK}_Si, S_i1^ei1,...,S_iN^eiN,{F}_K
α^x
α^x
α^y1,E_k1(S_KM1{α^y1,α^x})
α^yN,E_kN(S_KMN{α^yN,α^x})
E_K1(S_Cli{α^x,α^y1}),S_i1^ei1R^ei1, E_k1(P_i)
E_KN(S_Cli{α^x,α^yN}),S_i1^eiNR^eiN, E_kN(P_i)
E_k1(S_i1R)
E_kN(S_iNR)

Fig. 7.12. DaSCE File Download with Multiple Key Managers.

client downloading the data from the cloud. To decrypt *F*, we need *K* that is encrypted with S_i. The S_i is encrypted with (e_i, n_i) received from *KM*. The client establishes the session key with the *KMs* and during the process both the client and the *KMs* authenticate each other through digital signatures. The process of key establishment and authentication is the same as discussed in Section 5.2. In the third step, after verifying the authenticity of the *KMs*, the client generates a random number *R* and encrypts it with the public key of the corresponding *KM*. The client then calculates $S_i^{ei}R^{ei}$ and sends it along with its own signature and encrypted P_i. We combine these steps to minimize the communication overhead. The *KM* after verifying the digital signature of the client decrypts P_i and checks whether the policy still holds or otherwise. If the policy is valid, then the *KM* decrypts $S_i^{ei}R^{ei}$ with the corresponding d_i to generate S_iR. The purpose of *R* is to mask the actual value of S_i. The *KM* encrypts S_iR with the session key, which is sent to the client.

It is noteworthy to mention that in FADE, S_iR is returned by applying ABE. However, in the DaSCE, we do not use ABE, instead session key is used to send S_iR to the legitimate user. Therefore, the access control is being managed by the aforementioned technique. The client after receiving S_iR extracts S_i from S_iR. It is important to remember that with multiple *KMs*, a share of S_i will be received from at least *k KMs*. Consequently *k* number of S_i*s* will be used to generate S_i. The client decrypts *K* and *IK* using S_i. It verifies the integrity of *F* using *IK* and decrypts *F* upon successful verification.

7.4.4. Policy Revocation

The same process of key establishment, as discussed in Section 5.2, is used for the policy revocation in DaSCE. The client encrypts P_i with the session key and sends to *KM*. The *KM* after performing decryption on P_i revokes the keys generated with P_i. The deleted keys include the private key d_i and associated prime numbers p_i and q_i. It also sends acknowledgement to the client.

When d_i associated with P_i is deleted, the corresponding S_i cannot be decrypted. This results in logical deletion of F as K cannot be decrypted without S_i. Therefore, we say that F is assuredly deleted. It is noteworthy that assured deletion does not correspond to the physical deletion of data. It is difficult to get the assurance of file deletion from the system outside the administrative control of data owner. For assured deletion we used the concept introduced in [7.34] and [7.35], where the inaccessibility of data is assured by deleting certain important information from the system. The DaSCE ensures the inaccessibility of the keys to make the data unrecoverable. Therefore, the main security property of file assured deletion is that even if a *KM* does not remove the key from its storage, the data files remain encrypted and unrecoverable. The concept of file assured deletion is also termed as self-destructing data in the literature. For details about file assured deletion, readers are encouraged to see [7.34] and [7.35].

To boost the level of trust in the proposed scheme, the key generation and management is not dependent on a single *KM*. Shamir's secret sharing scheme is applied to counter any malicious *KM*. Any malicious *KM* cannot get hold of S_i independently. At least k number of *KMs* needs to be compromised in order to get access of enough d_i*'s* that can be used to decrypt S_i. It is also noteworthy that for decryption process S_i is sent to *KM*. However, S_i is not sent in plain as discussed in Section 5.3. The S_i is masked by multiplication with R. Therefore, even if malicious *KM* keeps the resultant decrypted information, the extraction of S_i will remain a challenge. Therefore, aforementioned case of malicious *KM* seems hard to be translated into successful attack. If we build a case of a malicious user that somehow has got hold of some other user's encrypted S_i, the malicious user has to go through the authentication process of at least k number of *KMs* to decrypt the S_i. We will see in Section 5.6 that *KMs* do not give access to the unauthorized users.

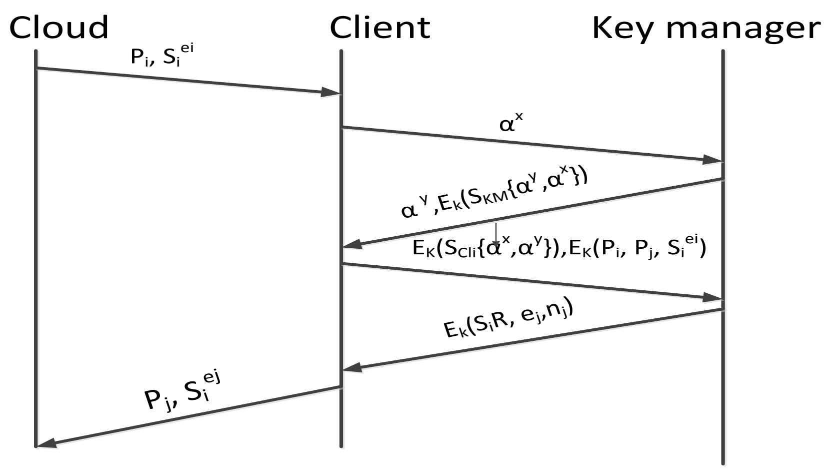

Cloud
Client
Key manager
P_i, S_i^{ei}
α^x
$\alpha^y, E_k(S_{KM}\{\alpha^y, \alpha^x\})$
$E_K(S_{Cli}\{\alpha^x, \alpha^y\}), E_K(P_i, P_j, S_i^{ei})$
$E_k(S_iR, e_j, n_j)$
P_j, S_i^{ej}

Fig. 7.13. DaSCE Policy Renewal.

7.4.5. Policy Renewal

The policy renewal does not involve any operation on *F*. The client downloads S_i and P_i; establishes session key with the *KM*; and sends P_i, $S_i^{ei}R^{ei}$, and P_j to *KM* by session key encryption. The *KM* decrypts $S_i^{ei}R^{ei}$ to obtain S_iR and generates new public/private key pair for P_j. Therefore, the *KM* sends S_iR and new public parameters (e_j, n_j) to the client. The client extracts S_i and re-encrypts it with (e_j, n_j). Finally, the client sends P_j and encrypted S_i to the cloud. Fig. 7.13 shows the process with single *KM*. The P_i in Fig. 7.12 is older policy file while P_j is the newer policy file.

7.4.6. Analysis of DaSCE through the HLPN

We use HLPN to verify that man-in-the-middle cannot forge the encryption keys exchanged between the client and the *KM*. If the intruder intercepts the messages, then the system would be able to identify the attack. The HLPN model for DaSCE is shown in Fig. 7.14.We assume an intruder between the client and the *KM* to check the behavior of the protocol in the attack scenario. The lines in Fig. 7.14 connecting (*c*1, *c*2) and (*c*3, *c*4) would be the information flow of X_a and X_b, respectively, if there is no intruder between the client and *KM*. Due to the space limitation and for simplicity we have not given the HLPN of the whole process.

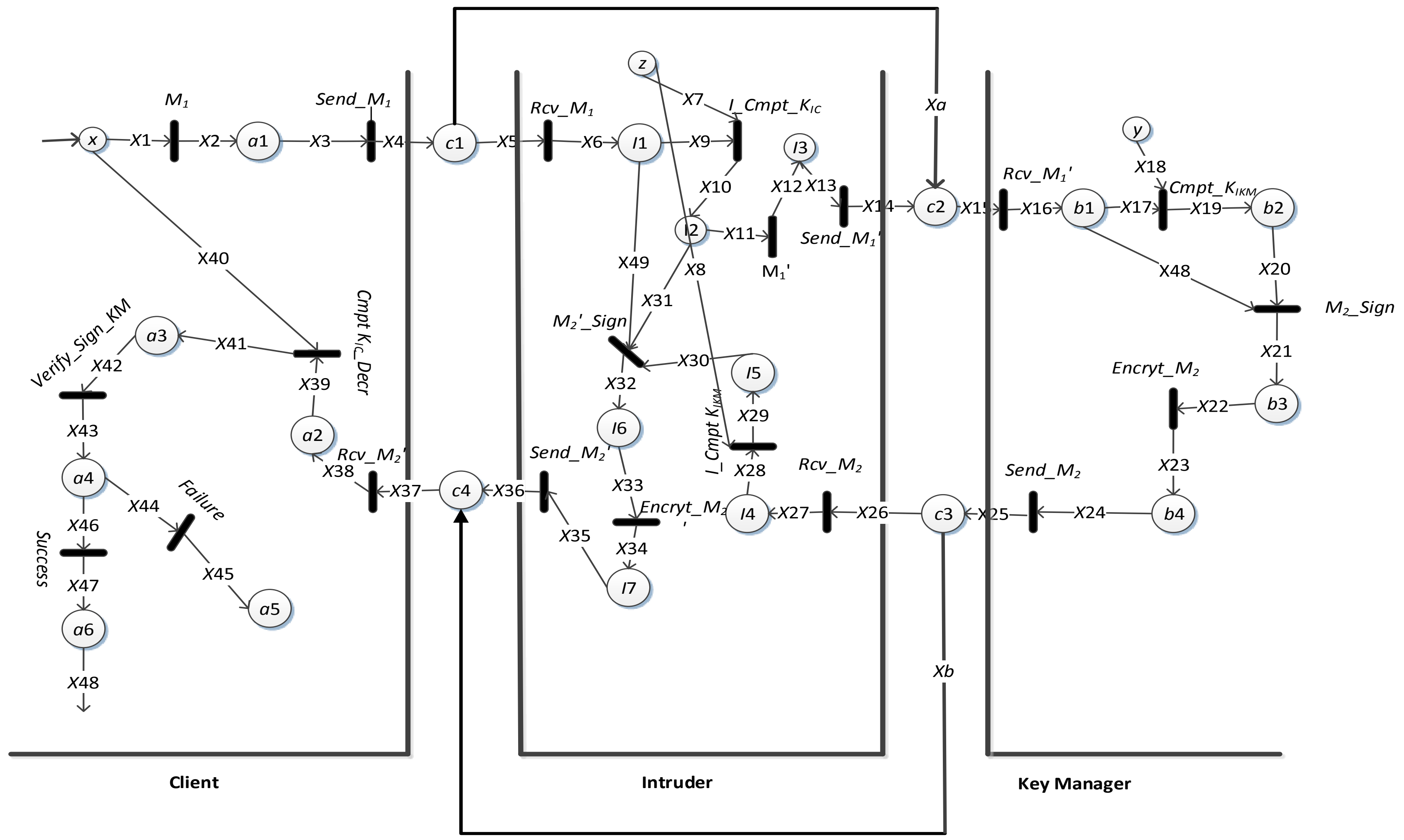


Fig. 7.14. HLPN for DaSCE.

The Fig. 7.14 only depicts the process of *KM* authentication. Nevertheless, the next step regarding authentication of client before exchanging keys will be the replication of the steps. Therefore, next step will have similar verification results. The associated data types and the mappings of places to data types are shown in Table 7.4 and Table 7.5, respectively. We assume an intruder between the client and the *KM* to check the behavior of the protocol in the attack scenario. The lines in Fig. 7.14 connecting (*c*1, *c*2) and (*c*3, *c*4) would be the information flow of X_a and X_b, respectively, if there is no intruder between the client and *KM*. Due to the space limitation and for simplicity we have not given the HLPN of the whole process.

Table 7.4. Data Types for HLPN of DaSCE

Types	**Description**
X	Big integer type random number for client
A	Big integer type number
Z	Big integer type random number for intruder
Y	Big integer type random number for key manager
M_1	Big integer type number representing α power x
M_1'	Big integer type number representing α power z
M_2	Big integer type number representing α power y
d_i	Private key of entity i from {*Cli*, *Clo*, *KM*}
e_i	Public key of entity i from {*Cli*, *Clo*, *KM*}
K_{IKM}	Session key between Intruder and Key Manager
K_{IC}	Session key between Intruder and Client
γ_s	$\{M_2, M_1'\}d_{KM}$ [M_2 and M_1' signed with d_{KM}].
γ_a	$\{\gamma_s\}K_{IKM}$ [γ_s encrypted with K_{IKM}]
M_2'	M_1',$(\gamma_a)K_{IC}$ [M_1' and γ_a encrypted with K_{IC}]
γ_i	$\{M_1', M_1\}d_i$ [$M_1'$and M_1 signed with d_i].
γ_b	$\{\gamma_i\}K_{IC}$ [γ_i encrypted with K_{IC}]
EM	Error Message (Message not coming from valid *KM*)

Table 7.5. Mapping of Data Types and Places for HLPN of DaSCE.

Types	**Description**	**Types**	**Description**
φ(x)	$\mathbb{P}(x)$	φ(b4)	$\mathbb{P}(M_2 \times \gamma_a)$
φ(a1)	$\mathbb{P}(M_1)$	φ(c3)	$\mathbb{P}(M_2 \times \gamma_a)$
φ(c1)	$\mathbb{P}(M_1)$	φ(I4)	$\mathbb{P}(M_2 \times \gamma_a)$
φ(I1)	$\mathbb{P}(M_1)$	φ(I5)	$\mathbb{P}(K_{IKM})$
φ(I2)	$\mathbb{P}(M_1' \times K_{IC})$	φ(I6)	$\mathbb{P}(M_1' \times d_i \times \gamma_i \times K_{IKM} \times K_{IC})$
φ(I3)	$\mathbb{P}(M_1')$	φ(I7)	$\mathbb{P}(M_1' \times \gamma_b)$
φ(c2)	$\mathbb{P}(M_1')$	φ(c4)	$\mathbb{P}(M_1' \times \gamma_b)$
φ(b1)	$\mathbb{P}(M_1')$	φ(a2)	$\mathbb{P}(M_1' \times \gamma_b)$
φ(b2)	$\mathbb{P}(M_2 \times K_{IKM})$	φ(a3)	$\mathbb{P}(K_{IC} \times \gamma_b)$
φ(b3)	$\mathbb{P}(M_2 \times d_{KM} \times \gamma_s \times K_{IKM})$	φ(a4)	$\mathbb{P}(K_{IC} \times \gamma_b \times e_{KM})$

The process starts with the client requiring an upload of data to the cloud. The client generates a random number *x*, calculates its parameters (as explained in Section 5.2), and sends to *KM*. However, the intruder intercepts the messages. The aforementioned process is carried out at transitions M_1, *Send_*M_1, and *Rcv_*M_1. The rules for these transitions are:

$$\boldsymbol{R}(M_1) = \forall\, x_1 \in X_1, \forall x_2 \in X_2 \mid x_2 \coloneqq pow(\alpha, x_1) \wedge \tag{7.13}$$

$$X_2' = X_2 \cup \{x_2\},$$

$$\boldsymbol{R}(Send\_M_1) = \forall\, x_3 \in X_3, \forall x_4 \in X_4 \mid x_4 \coloneqq x_3 \wedge \tag{7.14}$$

$$X_4' = X_4 \cup \{x_4\},$$

$$\boldsymbol{R}(Rec\_M_1) = \forall\, x_5 \in X_5, \forall x_6 \in X_6 \mid x_6 \coloneqq x_5 \wedge \tag{7.15}$$

$$X_6' = X_6 \cup \{x_6\}.$$

The transition *I_Cmpt_K_{IC}* is fired when the intruder successfully intercepts the message that is originated for the *KM*. The intruder generates its own random number *z* and calculates a key between the client and itself. The intruder also generates fake message for the *KM* at transition *M_1'*and sends it to the *KM* through transition *Send_ M_1'*. Rules (7.16) – (7.19) are mapped to following transitions.

$$\boldsymbol{R}(I\_Cmpt\_K_{IC}) = \forall\, x_7 \in X_7, \forall x_9 \in X_9, \forall x_{10} \in X_{10} |\, x_{10}[1] \coloneqq pow(\alpha, x_7) \wedge \quad (7.16)$$

$$x_{10}[2] \coloneqq pow(x_9, x_7) \wedge$$

$$X'_{10} = X_{10} \cup \{x_{10}[1], x_{10}[2]\},$$

$$\boldsymbol{R}(R\_M_1{}') = \forall\, x_{11} \in X_{11}, \forall x_{12} \in X_{12} |\, x_{12} \coloneqq x_{11} \wedge X'_{12} = X_{12} \cup \{x_{12}\}, \quad (7.17)$$

$$\boldsymbol{R}(Send\_M_1{}') = \forall\, x_{13} \in X_{13}, \forall x_{14} \in X_{14} |\, x_{14} \coloneqq x_{13} \wedge \quad (7.18)$$

$$X'_{14} = X_{14} \cup \{x_{14}\},$$

$$\boldsymbol{R}(Rec\_M_1{}') = \forall\, x_{15} \in X_{15}, \forall x_{16} \in X_{16} |\, x_{16} \coloneqq x_{15} \wedge \quad (7.19)$$

$$X'_{16} = X_{16} \cup \{x_{16}\}.$$

The *KM* assuming that the message comes from the client, calculates the session key by the parameters sent by the intruder. The *KM* also signs the received and generated parameters by the private key and sends to the client that is actually received by intruder. Following transitions and rules correspond to the explained steps.

$$R(Compt\_K_{IKM}) = \forall\, x_{17} \in X_{17}, \forall x_{18} \in X_{18}, \forall x_{19} \in X_{19} | x_{19}[1] \coloneqq pow(\alpha, x_{18}) \wedge \quad (7.20)$$

$$x_{19}[2] \coloneqq pow(x_{17}, x_{18}) \wedge$$

$$X'_{19} = X_{19} \cup \{x_{19}[1], x_{19}[2]\},$$

$$\boldsymbol{R}(M_2\_Sign) = \forall\, x_{20} \in X_{20}, \forall x_{21} \in X_{21}, \forall x_{48} \in X_{48} | x_{21}[1] \coloneqq x_{20}[1] \quad (7.21)$$

$$\wedge\, x_{21}[2] \coloneqq sign(x_{20}[1], x_{48}) \wedge x_{21}[3] \coloneqq x_{20}[2] \wedge$$

$$X'_{21} = X_{21} \cup \{x_{21}[1], x_{21}[2], x_{21}[3]\},$$

$$\boldsymbol{R}(Encryt\_M_2) = \forall\, x_{22} \in X_{22}, \forall x_{23} \in X_{23} | x_{23}[1] \coloneqq x_{22}[1] \wedge \quad (7.22)$$
$$x_{23}[2] \coloneqq encrypt(x_{22}[2], x_{22}[3]) \wedge X'_{23} = X_{23} \cup \{x_{23}[1], x_{23}[2], x_{23}[3]\},$$

$$\boldsymbol{R}(Send\_M_2) = \forall\, x_{24} \in X_{24}, \forall x_{25} \in X_{25} | x_{25} \coloneqq x_{24} \wedge$$
$$X'_{25} = X_{25} \cup \{x_{25}\}, \quad (7.23)$$

$$\boldsymbol{R}(Rcv\_M_2) = \forall\, x_{26} \in X_{26}, \forall x_{27} \in X_{27} |\, x_{27} \coloneqq x_{26} \wedge$$
$$X'_{27} = X_{27} \cup \{x_{27}\}. \quad (7.24)$$

After receiving the message from the *KM*, the intruder generates the session key between the *KM* and itself. At this stage, the intruder sets up the keys with both the client and the *KM*. The intruder prepares a response for the client and sends the prepared response. The response includes the signed parameters. The intruder uses its private key for the signing purpose. The client accepts the response thinking it to be from *KM*. The following rules highlight the process.

$$\boldsymbol{R}(I\_Cmpt\_K_{IKM}) = \forall\, x_{28} \in X_{28}, \forall x_{29} \in X_{29}, \forall x_8 \in X_8 |\, x_{29} \coloneqq pow(x_{28}[1], x_8) \wedge \quad (7.25)$$
$$X'_{29} = X_{29} \cup \{x_{29}\},$$

$$\boldsymbol{R}(M_2{}'\_Sign) = \forall\, x_{30} \in X_{30}, \forall x_{31} \in X_{31}, \forall x_{32} \in X_{32}, \forall x_{49} \in X_{49} | \quad (7.26)$$
$$x_{32}[1] \coloneqq x_{31}[1] \wedge x_{32}[2] \coloneqq sign(x_{31}[1], x_{49}) \wedge x_{32}[3] \coloneqq x_{30} \wedge x_{32}[4] \coloneqq x_{31}[2] \wedge$$
$$X'_{32} = X_{32} \cup \{x_{32}[1], x_{32}[2], x_{32}[3], x_{32}[4]\},$$

$$\boldsymbol{R}(Encr\_M_2{}') = \forall\, x_{33} \in X_{33}, \forall x_{34} \in X_{34} |\, x_{34}[1] \coloneqq x_{33}[1] \wedge \quad (7.27)$$
$$x_{34}[2] \coloneqq encrypt(x_{33}[2], x_{33}[4]) \wedge$$
$$X'_{34} = X_{34} \cup \{x_{34}[1], x_{34}[2]\},$$

$$\boldsymbol{R}(Send\_M_2{}') = \forall\, x_{35} \in X_{35}, \forall x_{36} \in X_{36} |\, x_{36} \coloneqq x_{35} \wedge \quad (7.28)$$
$$X'_{36} = X_{36} \cup \{x_{36}\},$$

$$\boldsymbol{R}(Rcv\_M_2{}') = \forall\, x_{37} \in X_{37}, \forall x_{38} \in X_{38} |\, x_{38} \coloneqq x_{37} \wedge \quad (7.29)$$

$$X'_{38} = X_{38} \cup \{x_{38}\}.$$

The client after receiving the response completes the process of generating the session key. However, the key generated is between the client and the intruder, instead of being between the client and the *KM*. Following this, the client decrypts the received parameters and verifies the digital signature over them. The verification is performed using a public key of the *KM* as the client is supposedly interacting with the *KM*. The verification mechanism gives the false result and the client terminates the process. However, if there is no intruder and the communication takes place between the client and *KM*, then valid signatures will result in information flow towards the place *a6* and communication will proceed. Following are the transitions and rules for aforesaid process at the client end.

$$\boldsymbol{R}(Cmpt\_K_{IC}\_Decr) = \forall\, x_{39} \in X_{39}, \forall x_{40} \in X_{40}, \forall x_{41} \in X_{41} | \quad (7.30)$$

$$x_{41}[1] \coloneqq pow(x_{38}[1], x_{40}) \wedge x_{41}[2] \coloneqq decrypt(x_{38}[2], x_{41}[1]) \wedge$$

$$X'_{41} = X_{41} \cup \{x_{41}[1], x_{41}[2]\},$$

$$\boldsymbol{R}(Ver\_Sign\_KM) = \forall\, x_{42} \in X_{42}, \forall x_{43} \in X_{43} |\, x_{43} \coloneqq verifysign(x_{42}[2]) \wedge \quad (7.31)$$

$$X'_{43} = X_{43} \cup \{x_{43}\}.$$

The following properties are verified using the SMT-Lib and Z3 solver.

- During communication, if state $I1$ (see Fig. 7.14, intruder side) is achieved (that means the intruder intercepts communication), then the control will terminate at state $a5$ which represents a failure to authenticate the KM and the process terminates. The property in CTL* is represented as $AG\ (I1 \rightarrow AF\ a5)$
- If there is no intruder and communication progresses on normal course (through lines X_a and X_b in Fig. 7.13), then the control will flow until it reaches $a6$, which represents

success for authentication and secure exchange of the required key. CTL* property is $AG\ (a1 \rightarrow AF(\neg I1 \cup a6))$

Both of the properties are verified in SMT-Lib using Z3 solver that approximately took 321 msec.

7.5. Implementation and Performance Evaluation

We used C# for implementing a working prototype of DaSCE. The .Net cryptographic packages were used for the involved cryptographic operations. Large prime numbers were handled by using the BigInt class. Policies were uploaded as a separate file to the cloud and the KM. The system consists of two servers (the cloud and the KM) and a client (work station). Multiple policies were combined using OR and/or AND operations. The policy and data files were not merged into a single file, to keep the policy renewal operation light weight. According to the processes described in Section 5, we also implemented the client side software functions, such as file upload, download, revocation, and renewal.

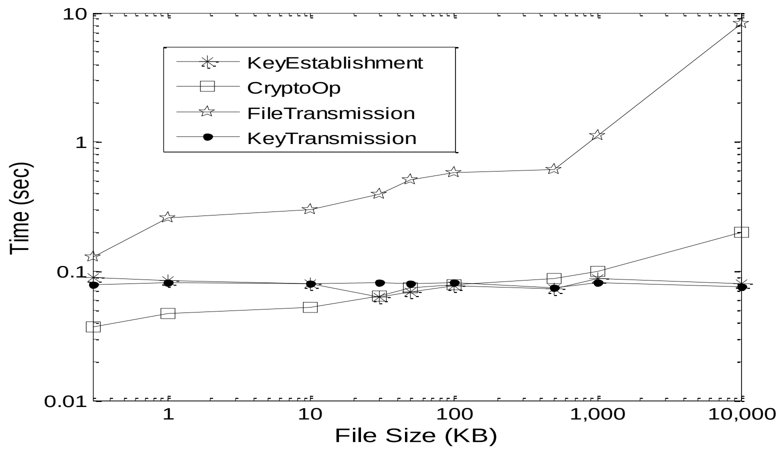
In our prototype, the client interacts with the *KM* (*s*) and the cloud for setting up the keys, and uploading/downloading data. The *KM* sets up the keys, revokes, and/or renews policies and manages the keys accordingly. We evaluated the DaSCE on the basis of: **(a)** Key(s) establishment time, **(b)** Key Transmission time, **(c)** File transmission time, and **(d)** Cryptographic operations time. It is noteworthy to mention that the time required for key establishment is the time for setting up a session key between the involved parties. The cryptographic operations time is the time taken by AES and MAC operations. Above given parameters collectively make up total file upload/download time. Moreover, the aforesaid parameters are evaluated using single *KM* and multiple *KMs*.

7.5.1. File Upload/Download with a Single Key Manager

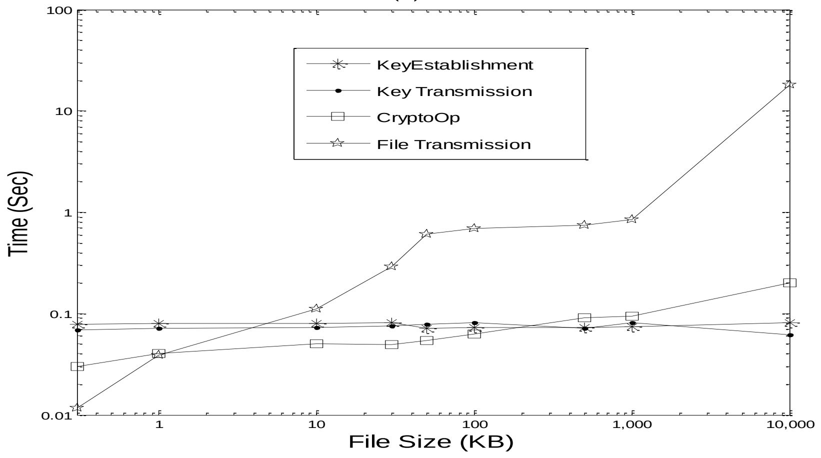
We used files of nine different sizes (0.3 KB, 1 KB, 10 KB, 30 KB, 50 KB, 100 KB, 500 KB, 1 MB, and 10 MB) to measure the time consumption in file upload and download process. The results are provided in Fig. 7.15. In general, the file transmission time increased with the increase in file size. However, in some cases the change in file transmission time was small that may be caused due to network conditions at various times. Nevertheless, file transmission time was dependent on the network. In file upload case, cryptographic operations time varied between 0.037 sec and 0.201 sec. The cryptographic operations time increased with the increase in the file size. In the case of 10 MB file, the cryptographic operations time makes 2.35% of total file upload time and 2.45% of file transmission time. The time for session key establishment almost remained constant (having slight changes). The largest time taken during the key establishment was noted to be 0.0898 sec that constituted 2.67% of the total upload time. The percentage for key establishment time was 2.39% for 10 MB file. Similarly, in case of file download operations the cryptographic operations time varied from 0.039 sec to 0.211 sec. The cryptographic operations time was dependent on the size of the file; therefore, it increased with the larger file size. However, it made lower percentage of total upload time and file transmission time. The key establishment time does not depend on the file size; therefore, it remains almost constant. Slight changes were possibly due to network transmission conditions. The DaSCE and FADE takes same amount of time for cryptographic operations. However, unlike FADE, we perform additional steps for key establishment in DaSCE that makes an additional overheads. Therefore, key establishment process increases the time consumption of DaSCE as compared to the protocols that run without establishing the session keys. It is noteworthy that the increase in time consumption upturns the security level for policy files, symmetric, and asymmetric keys used in the DaSCE. In the following section we will see the impact of key establishment time with increase in number of KMs.

7.5.2. File Upload/Download with Multiple Key Managers

We evaluated the performance of DaSCE by using multiple *KMs*. The file sizes we used were 0.3 KB, 1 KB, 10 KB, 50 KB, 100 KB, 500 KB, and 1MB. The number of *KMs* used was one, three, five, seven, fifteen, 25, and 50. Fig. 7.16 revealed the key establishment time and the cryptographic operation time for the aforementioned files sizes and the *KMs*. The key establishment time increased with the increase in the number of *KMs*. This is because the client had to complete all the message passing steps necessary to establish the key with all the *KMs*.



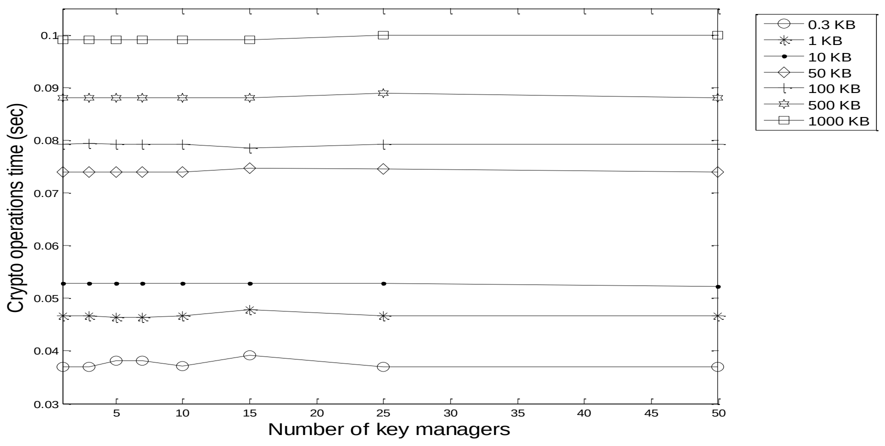
(a)



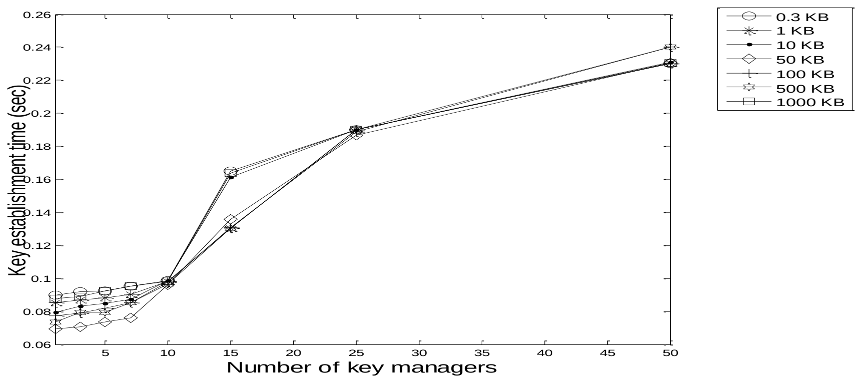
(b)

Fig. 7.15 Performance of File Uploads and Downloads Operations for DaSCE.

The key establishment time varies between 0.069 (single KM) seconds and 0.24 seconds (50 KMs). It must be noted that there was slight increase in the key establishment up to ten KMs. However, with higher number of KMs the increase followed a higher trend. As discussed earlier, the increase in time consumption due to key establishment augments the security level. Therefore, we say that user has to select the number of KMs judicially. A balance between tolerate able time consumption and security level in needed while deciding the number of KMs. In the coming discussion we will also see that the key establishment time constitutes low percentage of total time.

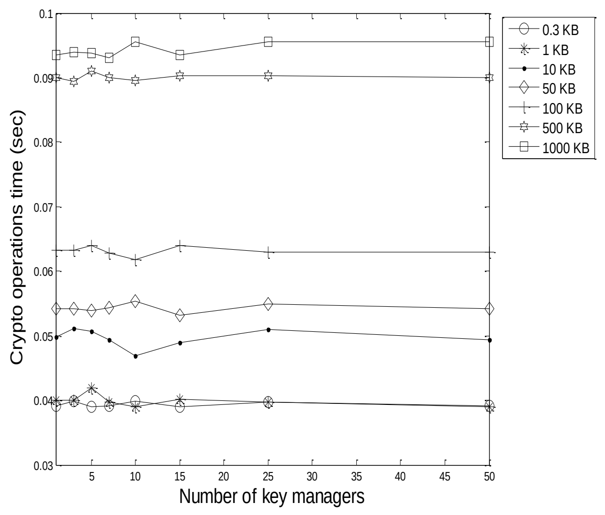


(a) Cryptographic operations

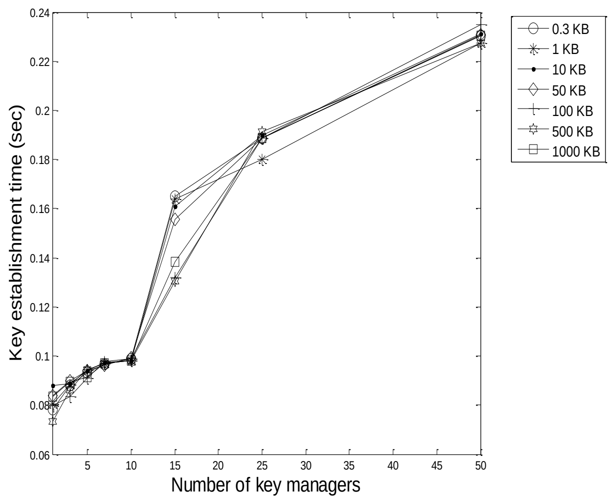


(b) Key establishment

Fig. 7.16 File Uploads with Multiple Key Managers.



(a) Cryptographic operations



(b) Key establishment
(c)

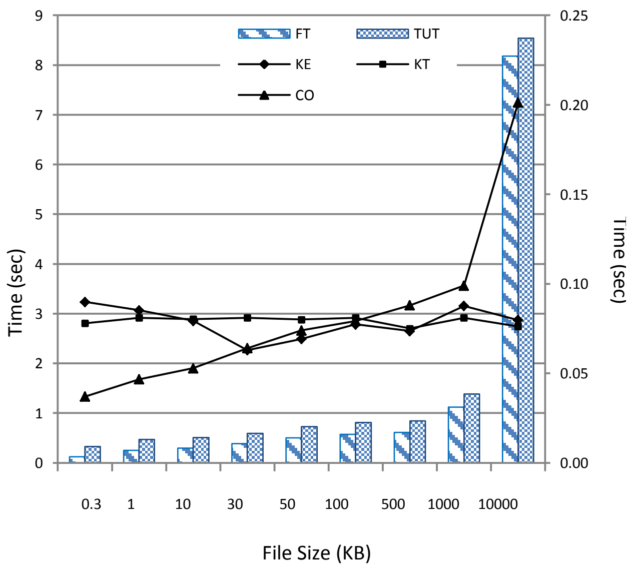
Fig. 7.17 File Download with Multiple Key Managers.

The cryptographic operation time remained constant for the file of same size as final symmetric encryption is done on client with generated keys (symmetric key, *K*). Fig. 7.17 depicts the key establishment time and cryptographic operation time taken by file download with multiple *KMs*.

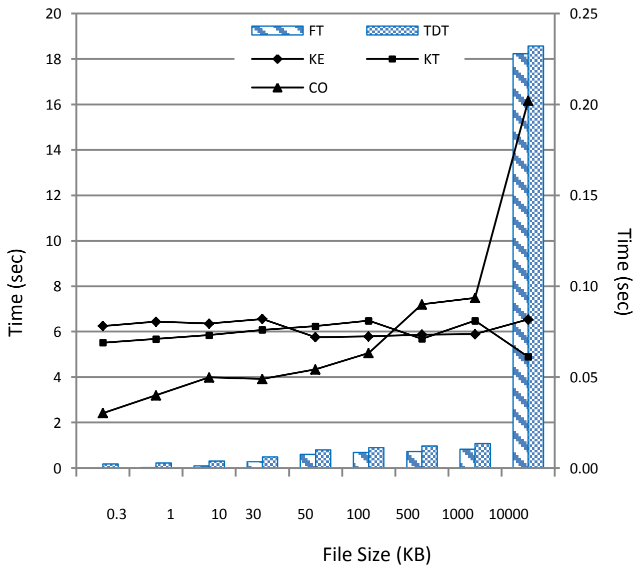
It must be noted that the key establishment constituted a low percentage of the total consumed time, see Fig. 7.17. Fig. 7.18 contains time comparisons of total upload/download time with key establishment and other constituent times for single key managers with different file sizes. It can be noted that as the amount of data increases, the percentage of key establishment time decreases to less significant number as compared to the total upload and download time, for the file (see Fig. 7.18). Therefore, an increase in number of key managers will increase the security as well as the time consumption due to key establishment.

7.5.3. Discussion

We present DaSCE that augments the security level by introducing additional steps for the key establishment process. Because of the mutual exclusion of communication events between the client and the KM, FADE fell short on issues of securing the keys and authentication of participating parties. The DaSCE resolves the aforesaid issues by introducing digital signature for the authentication and session key establishment process before any other exchange takes place. Moreover, the concept of “assured file deletion” is used to make the file inaccessible or unrecoverable by deleting important information (d_i). Comparing the performance of FADE and DaSCE, FADE has less performance overheads as compared to DaSCE. However, unlike FADE, the DaSCE provides high security standards and does not compromise the keys under man-in-the-middle attack. It is noteworthy that DaSCE does not introduce substantial performance and monetary overhead that can lead to higher management cost. However, as compared to FADE, the performance overhead of DaSCE are slightly higher because of the supplementary steps taken to increase the level of security for the keys that upturns the security level for policy files, symmetric, and asymmetric keys used in the DaSCE.



(a) Upload time



(b) Download time

Fig. 7.18 Total Upload/Download Time vs Key Establishment Time. KE = Key Establishment Time, Kt= Key Transmission Time, CO = Cryptographic Operations Time, FT= File Transmission Time, TUT= Total Upload Time, TDT= Total Download Time.

7.6. References

[7.1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Ktaz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoics, and M. Zaharia, “A View of Cloud Computing,” *Communications of the ACM*, Vol. 53, No. 4, 2010, pp. 50-58.

[7.2] M. S. Blumenthal, “Is Security Lost in the Clouds?” *Communications and Strategies,* No. 81, 2011, pp. 69-86.

[7.3] C.Cachinand M.Schunter, "A cloud you can trust," *IEEE Spectrum*, Vol. 48, No. 12, 2011,pp. 28-51.

[7.4] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, "Controlling data in the cloud: outsourcing computation without outsourcing control," *In Proceedings of the ACM workshop on Cloud computing security*,pp. 85-90, 2009.

[7.5] C. Cremers, "The Scyther Tool: Verification, falsification, and analysis of security protocols." *In Computer Aided Verification, Springer Berlin Heidelberg*, 2008, pp. 414-418.

[7.6] Cloud Security Alliance https://downloads.cloudsecurityalliance.org/initiatives/cdg/CSA_CCAQIS_Survey.pdf (accessed March 24, 2013).

[7.7] D.R. Dams, “Flat fragments of CTL and CTL*: spreading the expressive and distinguishing powers,” *Logic Journal of IGPL*, Vol. 17, No. 1, 1999, pp. 55-78.

[7.8] J. Desel and J.Esparza, “Free Choice Petri Nets,” *Cambridge Tracts in Theoretical Computer Science,* Vol. 40, Cambridge, UK: Cambridge Univ. Press, 1995.

[7.9] W. Diffie, P. C. V. Oorschot, and M. J. Wiener, “Authentication and authenticated key exchanges,” *Designs, Codes and Cryptography,* Vol. 2, No. 2, 1992, pp. 107-125.

[7.10] T. Dillon, C. Wu, and E. Chang, “Cloud Computing: Issues and Challenges,” *In proceedings of 24th International Conference on Advanced Information Networking and Applications*, pp. 27-33, 2010.

[7.11] N. En and N. Srensson, “An extensible SAT-solver,” *Lecture Notes in Computer Science*, vol. 2919, Springer, 2003, pp. 502-518.

[7.12] C P. Gomes, H. Kautz, A. Sabharwal, and B. Selman, “Satisfia-bility solvers,” *In Handbook of Knowledge Representation, Elsevier*, 2007.

[7.13] A. N. Khan, M. L. Mat Kiah, S. U. Khan, and S. A. Madani, “Towards secure mobile cloud computing: a survey,” *Future Generation Computer Systems*, Vol. 29, No. 5, 2013, pp. 1278-1299.

[7.14] A. Juels and A. Opera, “New approaches to security and availability for cloud data,” *Communications of the ACM*, Vol. 56, No. 2, 2013, pp. 64-73.

[7.15] S. Kamara and K. Lauter, “Cryptographic cloud storage,” *Financial Cryptography and Data Security,* Springer Berlin Heidelberg, 2010, pp. 136-149.

[7.16] M. Kaufman,”Data security in the world of cloud computing,” *IEEE Security and Privacy*, Vol. 7, No. 4, 2009, pp. 61-64.

[7.17] K. M. Khan, and Q. Malluhi, "Establishing trust in cloud computing," *IT professional*, Vol. 12, No. 5, 2010,pp. 20-27.

[7.18] H. Lin and W. Tzeng, “A secure decentralized erasure code for distributed network storage,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 11, Nov. 2010, pp. 1586-1594.

[7.19] H. Lin and W. Tzeng, “A secure erasure code-based cloud storage system with secure data forwarding,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 6, June 2012, pp. 995-1003.

[7.20] M. Maidl, “The common fragment of CTL and LTL,” *IEEE symposium on foundations of computer science*, pp. 643-652, 2000.

[7.21] S. U. R. Malik, S. K. Srinivasan, S. U. Khan, and L. Wang, “A Methodology for OSPF Routing Protocol Verification,” *12th International Conference on Scalable Computing and Communications (ScalCom)*, Changzhou, China, Dec. 2012.

[7.22] L. Moura and N. Bjrner, “Satisfiability Modulo Theories: An appetizer,” *Lecture Notes in Computer Science*, Vol. 5902, Springer, 2009, pp. 23-36.

[7.23] M. Mowbray, and S. Pearson, "A client-based privacy manager for cloud computing," *In Proceedings of the Fourth International (ICST) Conference on COMmunication System softWAre and middleware, ACM*, p. 5, 2009.

[7.24] T. Murata, “Petri Nets: Properties, Analysis and Applications,” *Proc. IEEE*, Vol. 77, No. 4, pp. 541-580, Apr. 1989.

[7.25] W. Reisig and G. Rozenberg, “Lectures on Petri Nets I: Basic Models,” *Lecture Notes in Com-puter Science*, Berlin: Springer-Verlag, Vol. 1491, 1998.

[7.26] A. Shamir, “How to Share a Secret,” *Comm. ACM*, Vol. 22, No. 11, Nov. 1979, pp. 612-613.

[7.27] D. Sun, G. Chang, L. Sun, and X. Wang, “Surveying and Analyzing Security, Privacy and Trust Issues in Cloud Computing Environments,” *Procedia Engineering*, Vol. 15, 2011, pp. 2852 – 2856.

[7.28] H. Takabi, J. B. D. Joshi, and G. J. Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Security and Privacy*, Vol. 8, No. 6, 2010,pp. 24-31.

[7.29] Y. Tang, P. P. Lee, J. C. S. Lui, and R. Perlman, "Secure Overlay Cloud Storage with Access Control and Assured Deletion," *IEEE Transactions on Dependable and Secure Computing*, Vol. 9, No. 6, Nov. 2012, pp. 903-916.

[7.30] A. Yun, C. Shi, and Y. Kim, "On protecting integrity and confidentiality of cryptographic file system for outscored storage," *Proceedings of 2009 ACM workshop on cloud computing security CCSA'09*, pp. 67-76, 2009.

[7.31] W. Jansen and T. Grance, "Guidelines on security and privacy in public cloud computing," *NIST special publication*, 800-144, 2011.

[7.32] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, Vol. 25, No. 6, 2009, pp. 599-616.

[7.33] A. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, "A survey of mobile cloud computing application models," *IEEE Communications Surveys and Tutorials*, 2013, 1-21.

[7.34] R. Perlman, "File system design with assured delete," *In Third IEEE International Security in Storage Workshop*, pp. 6, 2005.

[7.35] R. Geambasu, T. Kohno, A. A. Levy, and H. M. Levy, "Vanish: Increasing Data Privacy with Self-Destructing Data," *In USENIX Security Symposium*, pp. 299-316. 2009.

8. CONCLUSIONS

Large scale computing systems are growing exponentially, such as cloud and DC, to accommodate the escalating demands of user and applications. Similarly, as the size and complexity of the DC increases the concerns related to reliability, energy consumption, security, availability, and performance are also growing. Google reported a 20% revenue loss due to a delay of 500msecs in response time and Amazon reported a sales decrease of 1% due to an additional response time of 100msecs. The said examples indicate the importance and impact of the slightest inaccuracy in large-scale computing systems. In the said perspective, the use of FMs for verifying the functionality and reliability of the systems is compulsory. In our research, we focused on the application of FMs tools and techniques to investigate about the reliability and correctness of different applications running in large scale computing systems.

In Chapter 3, we have studied and analyzed three state-of-the-art VM-based open source cloud management platforms: **(a)** Eucalyptus, **(b)** Open Nebula, and **(c)** Nimbus. To model the systems with the advantage of providing a firm mathematical representation and to analyze the structural and behavioral properties, we used High-Level Petri Nets (HLPN). Moreover, the models are verified using SMT-Lib and Z3 solver. We used Model Checking approach to verify our models. Several properties are specified (using the specification and details available in documentation) and if the models satisfy those properties, then the model is declared correct. The verification results revealed that the models are correct and feasible as the numbers of Virtual Machines (VM) grow. This paper provides an in-depth formal analysis, modeling, and verification of the systems that will be helpful for the research community to further explore and understand the systems. Moreover, the paper also provides a strong foundation for new researchers to apprehend the meticulous knowledge of the systems. In future, we will analyze, model, and verify some other cloud management platforms, such as OpenStack, oVirt, and ECP. Moreover, we will also perform a detailed feasibility analysis of the aforesaid platforms under different SLA constraints.

OSPF is one of the most widely used IGP over the Internet today. The primary goal of OSPF is to provide fast convergence and load balancing to the network. In Chapter 4, we provide the intra area convergence time analysis of OSPF based on the: **(a)** DR, **(b)** cascading failures, and **(c)** topology, on a broadcast and NBMA segments. We simulated the detailed implementation of OSPF protocol built on the specifications available in RFC 2328. The BRITE topology generator was used for the interconnections among the routers to get more realistic results. The results from our simulation revealed that the convergence time of an area depends significantly on: **(a)** the number of DRs, **(b)** the placement of DRs, **(c)** interconnection amongst the routers, and **(d)** the number of routers, in a topology. Moreover, the results also exposed the fact that having more DR in an area can improve the convergence time of the specified segment. However, the overall convergence time of the area will decrease.

In Chapter 5, we modeled DC as a CPS to capture the thermal evolution and dynamics presented by DC components. We modeled DC as a Cyber Physical System (CPS) to capture the dynamics and evolution of the thermal properties exhibited by the DC. All software aspects, such as scheduling, load balancing, and all the computations performed by the devices were considered as the “Cyber” component. The supported infrastructure, such as servers, switches, and power supplies were modeled as the “Physical” component of the CPS. We modeled the heat dissipation of the major components of DC, such as servers and switches, and utilized the information to propose a thermal aware scheduling approach. Our proposed strategy was testified and demonstrated by executing on a real DC workload having more than 22,000 jobs, obtained from the CCR, State University of New York, at Buffalo. Moreover, we also performed a comparison of our proposed strategy with three existing strategies, FCFS, TASA, and GA-based. Furthermore, we performed formal analysis, modeling, and verification using HLPN, SMT-Lib, and Z3 solver. We investigated the impact of all of the scheduling approaches on the overall thermal signature and thermal uniformity among the pods within a DC. Our analysis revealed that the scheduling heuristics exhibit non-uniformity in thermal signatures among the pods. Such uneven thermal signatures lead to hotspots within the data center. The finding and results from the analysis were used to mitigate the ambient effect caused by the job allocation. The simulation results revealed that our proposed strategy maintains better thermal balance within the pods of DCs as compared to the other approaches. The formal verification performed using SMT-Lib and Z3 solver, matches the simulation results, where hotspots were identified in all of the studied approaches.

Formal analysis of routing protocols is compulsory for a secure and efficient performance of modern large scale networks. In the said perspective, in Chapter 6, we proposed a novel method to verify the properties of OSPF protocol using delay information of the routers. We have verified the protocol in two parts: **(a)** content verification and **(b)** route verification. For **(a)**, we verify the property that the LSDB for all the routers in an area must be identical. For **(b)**, we uses delay information to order the events and then verify if the events are occurring in the same order. The aforementioned properties are verified using SMT-LIB and Z3 solver. We simulated the detailed implementation of OSPF and BRITE topology generator was used for the generation of realistic topological interconnections. The proposed method can scale up the verification by reducing the state space and narrowing it down to a single parameter.

In Chapter 7, we proposed the DaSCE protocol, a cloud storage security system that provide key management, access control, and file assured deletion. Assured deletion was based on policies associated with the data file uploaded to the cloud. On the revocation of policies, access keys were deleted by the *KMs* that result in halting of the access to the data. Therefore, the files were logically deleted from the cloud. The key management was accomplished using (*k*, *n*) threshold secret sharing mechanism. We modeled and analyzed FADE. The analysis highlighted some issues in key management of FADE. The DaSCE improved the key management and authentication processes. The working of the DaSCE protocol was formally analyzed using HLPN, SMT-Lib, and Z3 solver. The performance of the DaSCE was evaluated based on the time consumption during file upload and download. The results revealed that the DaSCE protocol can be practically used for clouds for security of outsourced data.