Availability Models for Synchronization Server Infrastructure

Carlos Melo*, Jamilson Dantas*, Jean Araujo*†, Paulo Maciel*, Leo Mendonça‡, Rodrigo Branchini‡ and Luiz Kawakami‡
*Informatics Center, Federal University of Pernambuco, Recife, Brazil
†Academic Unit of Garanhuns, Federal Rural University of Pernambuco, Garanhuns, Brazil
‡Motorola Mobility, Jaguariúna, Brazil
{casm3, jrd, jcta, prmm}@cin.ufpe.br*, jean.teixeira@ufrpe.br†,{mxkp78, branchini, luizkawakami}@motorola.com‡

***Abstract*—Users of computer systems wish to keep their personal data safe, updated, fair and accessible by other terminals, like personal computers, smart phones, portable consoles and PDAs. To perform these activities, one technology has become popular in our daily lives: data synchronization. Companies that provide this kind of service must do it with the greatest availability possible since their clients need their data to be available whenever they want to access it, and their customers in the legal field must avoid financial losses through SLA contract breaches. This paper presents hierarchical models for evaluating the availability of a data synchronization server infrastructure. The results show an availability of 98.82% for the proposed architecture, which means an annual downtime of 103 hours, this is more than 4 days of unavailability, where users cannot perform data synchronization.**

***Keywords*—Data Synchronization, Availability Models, Cloud Computing, Hierarchical Modelling.**

I. Introduction

In computer systems, synchronization is related to maintaining data consistency between distributed devices, such as cell phones, personal computers, portable consoles and PDAs [1]. Keeping personal data safe, updated, fair and accessible by other terminals has become something of great interest among gadget users and companies that provide this kind of service and want to avoid financial losses through a service-level agreement (SLA) breach. The data synchronization model can be expressed as a group of devices connected to each other through the Internet, where each one of these devices is responsible for keeping a copy or replica of the data that belongs to a user, a group of users or an organization [2].

Techniques for making the synchronization of data more automated, secure and efficient emerged shortly after the beginning of the race for personal computer development. In the 1980s, an approach called "pessimist" was the most popular, which consists of the designation of a replica as responsible for always keeping the remaining the file copies up to date [3]. If the main replica was lost through corruption or storage problems, an algorithm performed a comparison between the remaining replicas and designated a new replica to become the main one. This approach works fine in local networks, where analyzing or keeping data is an inexpensive task, but it becomes impractical if we need to realize synchronization between a significant number of distributed devices that are connected to the Internet.

In contrast to pessimist approaches, the "optimistic" ones are on the rise, transposing the distance between devices and their networks and improving the data availability by not depending on a single primary copy. Updates are applied simultaneously in the background for all copies of the same data, which allows for a virtually unlimited number of replicas and the detection of conflicts right after they happen [3]. However, its biggest advantage may be interpreted as a disadvantage in some aspects. In keeping data updated among a large number of sources, some issues may occur such as competition and, sometimes, corruption. These problems may be mitigated using tools or frameworks developed to facilitate data synchronization between devices with a reduction of the occurrence of conflicts; this is also applicable to devices with low processing power and bandwidth limitations, such as mobile devices. A well-known specification for the development of these tools is the Synchronization Markup Language or SyncML.

Some related works, such as [4] and [5], develop and evaluate synchronizers based on SyncML specifications in different scenarios. In [5], improvements to SyncML are proposed in order to optimize its performance in data synchronization, while [4] proposes the use of a synchronizer in the sharing of information and data about Chinese medicine. Meanwhile, in [1], Stochastic Petri Nets are used to evaluate the performance of SyncML as a whole and to determine system bottlenecks and places where improvements can be made.

Our approach differs from the others by proposing hierarchical models for evaluating the availability of a synchronization server infrastructure based on the SyncML framework. The main research contributions are subsidies and artifacts to people or organizations that plan to offer or acquire this type of service and want to host them on cloud computing platforms. By knowing an architecture's availability beforehand, the cloud computing infrastructure providers may study ways to upgrade it, provide the service with greater availability and attend to pre-established SLAs, thereby avoiding financial losses.

The remaining of the paper is organized as follows. Section II shows the background used as a basis for the development of this study; Section III shows the proposed architecture for our study; Section IV presents the availability models proposed in this research. Section V provides our case of study. Finally, section VI presents the final remarks and future works.

II. Background

This section describes the fundamental concepts for understanding this paper.

A. Synchronization Markup Language (SyncML)

SyncML is an open standard for data synchronization between Internet-connected devices. Its an expandable standard and has high viability for both: market applications and to academy. Its description is entirely based on eXtensible Markup Language (XML), which characterizes one of its main features: the exchange of messages in XML format [1]. This makes SyncML a cross-platform protocol with wide use, since XML is a stable standard, recognized and interpreted by a number computer systems.

The SyncML appeared in 2000 through an initiative formed by major companies in the information and communications technology sector, such as Ericsson, IBM and Nokia [6]. These companies aimed to solve a common problem at the time: the lack of interoperability between mobile devices, applications and servers.

Today, SyncML is a standard owned by the Open Mobile Alliance (OMA), a consortium established by large companies, including the ones that created SyncML. The SyncML becomes one product with a Device Manager (DM), also owned by the consortium, creating the OMA Data Synchronization and Device Management. In its latest version 2.0, has become one of the most used protocols by manufacturers of mobile devices for remote management and software updates delivery [7].

B. Sensitivity Analysis

Sensitivity analysis is a method to determine the most influential factors in a system [8] [9]. It is able to detect the most critical system components where one can or should apply some technique that can improve the results for the metrics of interest.

There are some techniques that can provide us with a sensitivity analysis of a system, such as design of experiments and the one used in this paper, percentage difference. The percentage difference consists in fixing a list of parameters while changing one parameter value at time, until all parameters have been analyzed. Equation 1 shows how the percentage difference can be made.

$$SI = (\frac{D_{max} - D_{min}}{D_{max}}), \quad (1)$$

where SI is the sensitivity index for the selected parameter, D_{max} is maximum value the parameter may assume and D_{min} is the minimum value of the parameter.

III. System Architecture

The proposed architecture consists of a cloud computing environment composed of two computers, as can been seen in Figure 1. The focus of this paper is in the service on the provider side, which means that anything that happens from the network connection until the client side will not be considered.

Cloud computing is a paradigm that consist in the dynamic allocation of virtualized resources of hardware and software with one goal: to provide all kinds of services through the Internet [10].

There are a number of platforms that can be used to provide cloud computing infrastructure. One of them is the Elastic Utility Computing Architecture Linking Your Programs to Useful Systems, or simply "Eucalyptus". Eucalyptus is an open-source platform that is able to provide its users the possibility of offering all kinds of services as well as hybrid and private infrastructures [11]. The main components of Eucalyptus are listed below:

- *Cloud Controller (CLC)*: Monitors the running instances and defines the clusters that will be used to provide the instances;
- *Cluster Controller (CC)*: Responsible for defining which physical machine belongs to a cluster. It checks information about available resources, possible schedules for each instance, and the requests made by the CLC;
- *Storage Controller (SC)*: Responsible for providing block storage for virtual machines. It is able to create and manage storage blocks and protects them from unauthorized access;
- *Walrus*: This is a service file that allows users to send and collect information and virtual machine images over the Internet, and its protocols enable the migration and the download of virtual machines;
- *Node Controller (NC)*: This is a component running on the nodes of the infrastructure. It connects directly to the operating system and the hypervisor running on this computer. It is responsible for maintaining the life cycle of virtual machines.

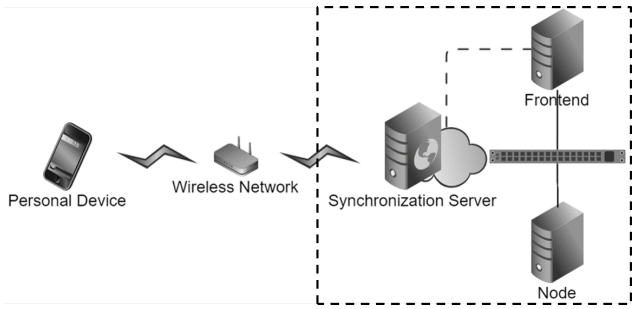

Fig. 1. System Architecture

The main components of the Eucalyptus cloud computing platform may be piled in two stacks, one representing the front-end computer and another representing the node. Figure 2 shows the high-level representation, which is characterized by creating an order that goes from the least critical component of the structure to the most critical one.

The Front-end's stack is composed of the following components running on the same machine: hardware (HW), operating system (OS), cluster controller (CC), cloud controller (CLC), storage controller (SC) and Walrus.

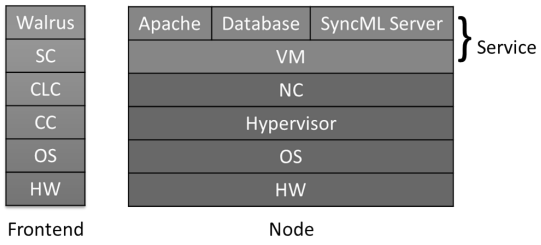

Fig. 2. Stack representation

The other Eucalyptus components are in the Node's stack: hardware (HW), operational system (OS), hypervisor, node controller (NC) and a "subsystem" representing the service. The service subsystem contains the virtual machine (VM), where all the applications responsible for the provision of the service are running: the Apache web server, the database and the SyncML synchronization server.

IV. Availability Models

Reliability block diagrams (RBDs) were used to represent the relationship between the Eucalyptus cloud components. The choice of RBDs was due to the absence of a priority among the components of this architecture, which implies that any component in a faulty state will affect the availability of the proposed service.

The architecture can be represented by a high-level RBD containing only two blocks: **Frontend** and **Node**, as shown in Figure 3. Each block has its own subsystem, earlier represented by the stacks in Figure 2 and now represented as RBDs in Figures 4 and 5. These RBDs are based in the ones proposed in [12] [13].

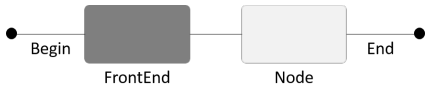

Fig. 3. Baseline Architecture

The general availability of the architecture can be given by Equation 2.

$$A_{System} = A_{Frontend} \times A_{Node} \tag{2}$$

In the Front-end architecture is represented by a pure series RBD, as shown in Figure 4. This subsystem consists of hardware, an operating system, and the following Eucalyptus components: CLC (Cloud Controller), CC (Cluster Controller), SC (Storage Controller) and Walrus [12]. The frontend's availability for the RBD is given by: $A_{Frontend}$ = $A_{HW} \times A_{OS} \times A_{CC} \times A_{CLC} \times A_{SC} \times A_{Walrus}$.

Figure 5 represents the node subsystem. This subsystem consists of hardware, an operating system, a hypervisor, a node controller and the service. The node's availability for the RBD is given by: A_{Node} = $A_{HW} \times A_{OS} \times A_{Hypervisor} \times A_{NC} \times A_{\textbf{service}}$. The service component is composed by a CTMC hierarchical model.

Unlike the Node and the Front-end, which are modelled with RBD, we opted for the use of continuous time Markov chains to model the service, which is composed of the SyncML synchronization server, the virtual machine, the Apache web server and the database. This choice was made with the purpose of using an immediate repair strategy that consists of the rebooting of the virtual machine. The CTMC with nine states can be seen in Figure 6.

The notation adopted to define each CTMC components resumes to (**U**) for when its **UP** and (**D**) for when its **DOWN**; the first character represents the state of the Virtual Machine, the second represents the database, the third one the Apache Webserver, and the fourth the SyncML component. The states are UUUU, UUUD, UDUD, UUDU, UDUU, UUDD, UDDU, UDDD and DDDD. The only state where the system is available is the UUUU state, since is the only one where all the components are in UP state.

The availability of the system is equal to the probability of the system being on UUUU state. In this state (Virtual Machine is up, Apache webserver is up, database is up, and the SyncML is up). Whenever the VM fails, the system becomes unavailable and can only return to the working state after rebooting. Upon a failure of any of the other components (Apache, db or SyncML), the system will go to an unavailability state representing the failure of the particular component. If another component fails, then the system goes to a state in which both components are failed. If the last component fails, the system goes to the state where all three components are not working, and the system requires a *reboot*.

V. Case Study

This section provides a case study to demonstrate how feasible are the models proposed.

A. Input parameters

In order to evaluate our models some input values are needed. Those values used for the components of the Eucalyptus platform and other software applications, such as failure and repair rate of database service, Apache and reboot time, are extracted from [12] [14] [15].

This scenario was chosen aiming a closer approximation to what happens in the real world; in this day and age more and more companies worldwide are migrating to cloud computing in order to offer their services over the Internet, and to do this, they need some assurance that their services will be delivered fairly and reliably; to comply with these guarantees there is the SLA contract. One of the major characteristic of a SLA is the establishment of the availability period for the infrastructure that will provide the service; and for the proposed case study this period corresponds to 365 days. The input values for frontend's RBD can be seen in Table I.

Table II shows input parameters for RBD Node.

Table III presents the input parameters for CTMC service model.

where **reboot** is the rate to reboot the virtual machine or instantiate a new one, which is equal to 52.17.

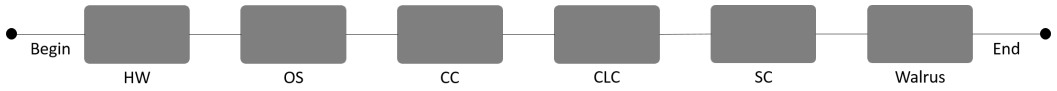

Fig. 4. Frontend

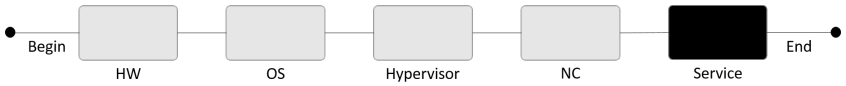

Fig. 5. Node

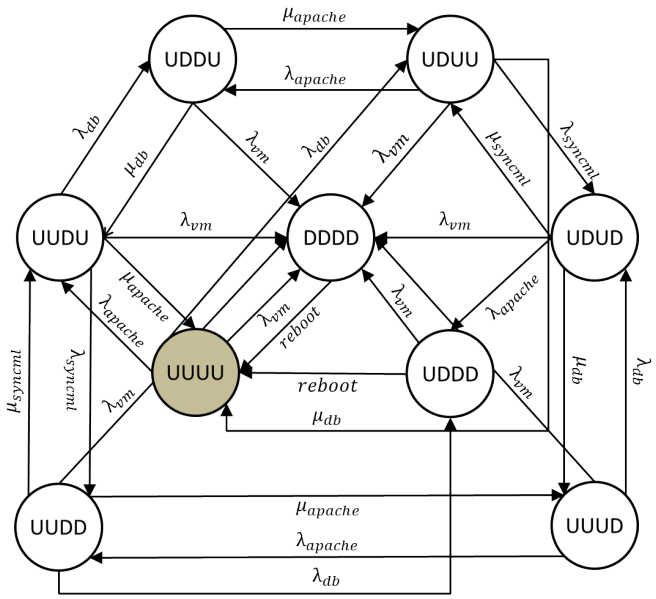

Fig. 6. CTMC Service

TABLE I. INPUT VALUES FOR FRONTEND'S RBD

Component	MTTF	MTTR
HW	8760 h	100 min
OS	2893 h	15 min
CLC	788.4 h	1 h
CC	788.4 h	1 h
SC	788.4 h	1 h
Walrus	788.4 h	1 h

B. Result analysis

The results of availability evaluation for the data synchronization server infrastructure are shown in Table IV, this evaluation considers both hardware and software components for the whole infrastructure.

With an availability of 98.82%, the system tends to be unavailable for 103.14 hours in one year, which correspond to more than four days each year where the users will be unable to synchronize their data. A question arises from these results: Which components may be affecting the system availability in this way? To answer this question, we can do a sensitivity analysis and find the main cause or the critical component of the system that can be improved in order to improve the overall system. The input parameters for the sensitivity analysis are shown in Table V. It is important to note that the acronym **db** is for the database, and **vm** is for the virtual machine.

TABLE II. INPUT PARAMETERS FOR NODE'S RBD

Component	MTTF	MTTR
HW	8760 h	100 min
OS	2893 h	15 min
Hypervisor	2990 h	1 h
NC	788.4 h	1 h

TABLE III. INPUT PARAMETERS FOR SERVICE CTMC

Component	λ (h^{-1})	μ (h^{-1})
SyncML	0.0012	1
Apache	0.0012	1
Database	0.0006	0.33
Virtual Machine	0.0003	reboot

TABLE IV. AVAILABILITY RESULT FOR THE ENTIRE SYSTEM

Metric	Value
Availability (%)	98.82
Availability (Number of 9's)	1.92
Annual Uptime (hours/year)	8656.86
Annual Downtime (hours/year)	103.14

The input values for sensitivity analysis are chosen based on the rates and times between failures and repairs for each system component. Each parameter varied between +50% and -50% of its original value, this measure was considered adequate to cover an upper and lower limit for the variation of architecture availability.

The percentage difference was the sensitivity analysis method used. The results of the sensitivity analysis can be seen in Table VI.

Some of the components ranked the highest in the sensitivity analysis are related directly or indirectly to the service and the node where the service is running and the range of values, and the rates of these components can significantly affect the availability of the service, as can be seen in λ_{db} and μ_{db}, failure and repair of the data base, which are service components. The variation of these rates are shown in Figures 7 and 8, where the continuous line represents the availability of the system and the dotted one represents what the availability of the system should be if we improve this rates.

If we manage to decrease the λ_{db}, we may greatly improve the availability of the system, but in the second case, if we increase the μ_{db} value, we will decrease the availability. This is a logical matter; with a higher rate, the system will be

TABLE V. INPUT PARAMETERS FOR SENSITIVITY ANALYSIS OF ARCHITECTURE

Parameter	Value
$MTTF_{nc}$, $MTTF_{walrus}$, $MTTF_{sc}$, $MTTF_{cc}$, $MTTF_{clc}$ (h)	788.4
$MTTF_{hypervisor}$ (h)	2990
λ_{syncml}, λ_{apache} (h^{-1})	$1/788.4$
λ_{db} (h^{-1})	$1/1440$
μ_{db} (h^{-1})	$1/3$
$reboot$ (h^{-1})	$1/0.019166$
λ_{vm} (h^{-1})	$1/2880$
$MTTR_{nc}$, $MTTR_{walrus}$, $MTTR_{sc}$, $MTTR_{cc}$, μ_{syncml}, μ_{apache} (h)	1
$MTTR_{os}$ (h)	0.25
$MTTF_{os}$ (h)	2893
$MTTR_{hw}$ (h)	1.66
$MTTF_{hw}$ (h)	8760

TABLE VI. SENSITIVITY INDEX OF SYNC SERVER ARCHITECTURE

Parameter	SS(A)
λ_{db}	2.46E-03
μ_{db}	2.07E-03
$MTTF_{nc}$, $MTTF_{walrus}$, $MTTF_{clc}$, $MTTF_{sc}$, $MTTF_{cc}$	1.69E-03
λ_{apache}, λ_{syncml}	1.50E-03
$MTTR_{nc}$, $MTTR_{walrus}$, $MTTR_{clc}$, $MTTR_{sc}$, $MTTR_{cc}$	1.13E-03
$MTTF_{hypervisor}$	4.46E-04
$MTTR_{hypervisor}$	2.97E-04
$MTTF_{hw}$	2.42E-04
$MTTF_{os}$	1.15E-04
$MTTR_{hw}$	1.01E-04
$MTTR_{os}$	7.68E-05
$reboot$	6.66E-06
λ_{vm}	4.28E-06

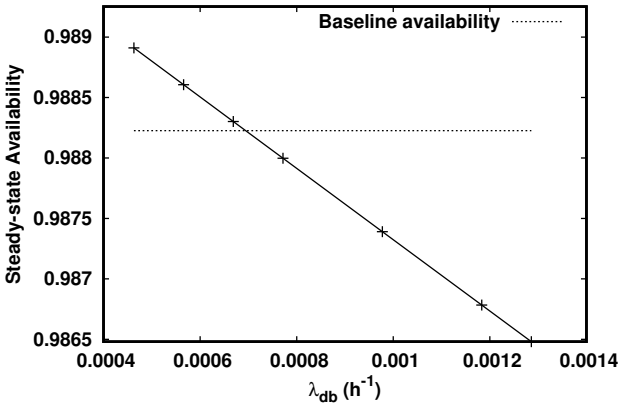

Fig. 7. Variation of λ_{db}

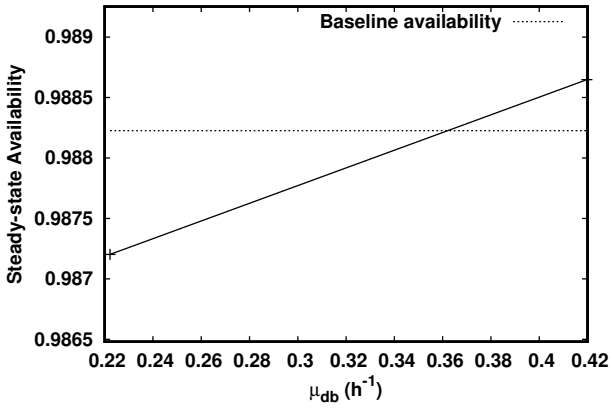

Fig. 8. Variation of μ_{db}

unavailable for more time than it should, and with a higher failure rate, the system is going to have more time available. However, this is only applicable for the components that really matter, unlike the $reboot$, which is the least critical component on the architecture, as can be seen in Figure 9.

Third and Fifth places in the sensitivity ranking are the Eucalyptus components at Front-end, each of these components are intrinsically related by the same time to failure and to repair. The Figure 10 shown the $MTTR_{walrus}$. With approximate 0.1% of variability, which is almost 9 hours of availability improvement in the one year scenario.

Sixth place in the ranking is the repair rate for SyncML and Apache, our sync and web servers respectively, Figure 11 shown the μ_{syncml} variability, the improvement is not as big as the previous results, but its a very important evaluation, since this component is determining in user's point of view on a possible fault state.

VI. FINAL REMARKS

This paper presented stochastic models, such as RBD and CTMC, to model the behavior and evaluate the availability of a data synchronization server supported by a private cloud computing platform. The results showed an availability of

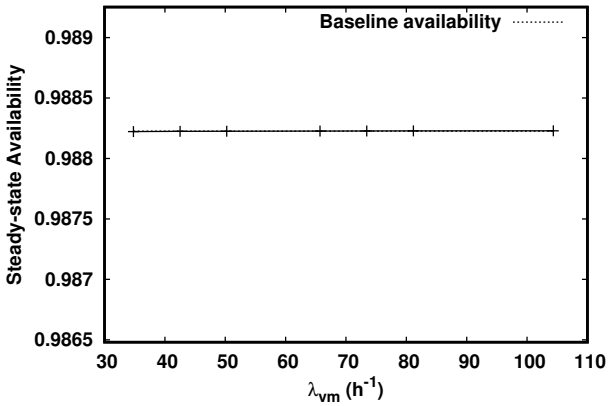

Fig. 9. Variation of $reboot$

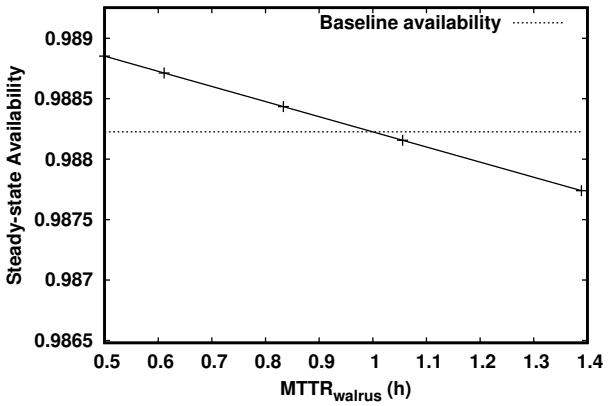

Fig. 10. Variation of Walrus MTTR

98.82% for the entire system, with an annual downtime of 103.14 hours. After that, we did a sensitivity analysis by using the percentage difference technique, aiming to identify the most critical component for the availability of the system. The results of the sensitivity analysis showed that the failure rate λ_{db} and repair rate μ_{db} are the most critical components, while the VM restoration, or $reboot$ is the least critical. In future work, we intend to implement redundancies in the system components, verifying the effect of these changes on the initial availability and annual downtime. Other possible future work is to evaluate the reliability of the system, by proposing new models to represent the system behavior without repairs rates, as well as another case of study.

VII. Acknowledgements

We would like to thank the Motorola Mobility for funding this research.

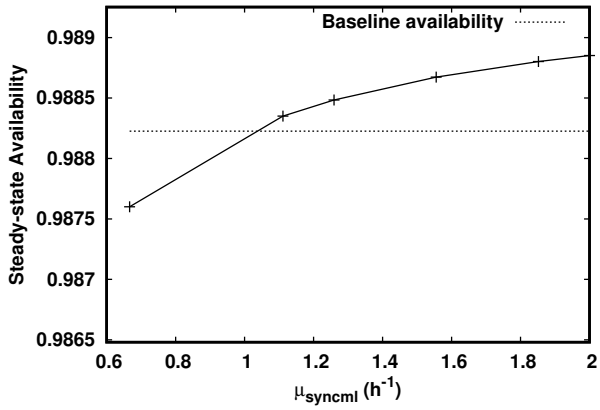

Fig. 11. Variation of μ_{syncml}

References

[1] B. Y. Lee, G. H. Lee, and Y. S. Kim, "Modeling and analysis of syncml server system using stochastic petri nets," in *Wireless and Mobile Communications, 2007. ICWMC '07. Third International Conference on*, March 2007, pp. 60–60.

[2] A. Traud, J. Nagler-Ihlein, F. Kargl, and M. Weber, "Cyclic data synchronization through reusing syncml," *2008 The Ninth International Conference on Mobile Data Management*, pp. 165–172, 2008.

[3] Y. Saito and M. Shapiro, "Optimistic replication," *ACM Computing Surveys*, vol. 37, pp. 42–81, 2005.

[4] C. Chun-lei and D. Jiang-qing, "Design and implementation of data synchronization in embedded tcm system based on syncml," *2010 International Conference on Computer Application and System Modeling*, vol. 10, pp. 619–622, 2010.

[5] Y. Tian and J.-P. Li, "Research and implementation of data synchronization with syncml," *2012 9th International Computer Conference on Wavelet Active Media Technology and Information Processing*, pp. 302–304, 2012.

[6] K. Dittrich and G. Duysters, "Networking as a means to strategy change: The case of open innovation in mobile telephony," *Journal of Product Innovation Management*, vol. 24, no. 6, pp. 510–521, 2007. [Online]. Available: http://dx.doi.org/10.1111/j.1540-5885.2007.00268.x

[7] Device management. Open Mobile Alliance. [Online]. Available: http://openmobilealliance.org/about-oma/work-program/device-management/

[8] P. M. Frank, *Introduction to Sensitivity Analysis*. Academic, 1978.

[9] R. Matos, J. Araujo, D. Oliveira, P. Maciel, and K. Trivedi, "Sensitivity analysis of a hierarchical model of mobile cloud computing," *Elsevier Journal Simulation Modelling Practice and Theory*, vol. 50, pp. 151–164, Jan. 2015.

[10] L. M. Vaqueiro, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: Towards a cloud definition," *Computer Communication Review*, vol. 39, pp. 50–55, 2009.

[11] Eucalyptus - the open source cloud platform. Eucalyptus Systems, Inc. [Online]. Available: http://open.eucalyptus.com/

[12] J. Dantas, R. Matos, J. Araujo, and P. Maciel, "Models for dependability analysis of cloud computing architectures for eucalyptus platform," *International Transactions on Systems Science and Applications*, vol. 8, pp. 13–25, 2012.

[13] ——, "Eucalyptus-based private clouds: availability modeling and comparison to the cost of a public cloud," *Computing*, vol. 97, no. 11, pp. 1121–1140, 2015. [Online]. Available: http://dx.doi.org/10.1007/s00607-015-0447-8

[14] I. Costa, J. Araujo, J. Dantas, E. Campos, F. A. Silva, and P. Maciel, "Availability evaluation and sensitivity analysis of a mobile backend-as-a-service platform," *Journal Quality and Reliability Engineering International*, 2015.

[15] E. Campos, R. Matos, A. Pereira, F. Vieira, and P. Maciel, "Stochastic modeling of auto scaling mechanism in private clouds for supporting performance tunning," in *Proceedings of the IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'15)*, Hong Kong, China, 2015.