Verifying Cloud Service Level Agreement

Lin Ye, Hongli Zhang, Jiantao Shi
School of Computer Science and Technology
Harbin Institute of Technology
Harbin, China
Email: {yelin, zhl, shijiantao}@pact518.hit.edu.cn

Xiaojiang Du
Dept. of Computer and Information Sciences
Temple University
Philadelphia, PA, USA
Email: dxj@ieee.org

***Abstract*—In this paper we study the important issue of verifying Service Level Agreement (SLA) in a semi-trusted (or untrusted) cloud. Cloud computing services promise elastic computing and storage resources in a pay-as-you-go way. A SLA between a cloud service provider (CSP) and a user is a contract which specifies the resources and performances that the cloud should offer. However, the CSP has the incentive to cheat on SLA, e.g., providing users with less CPU and memory resources than that specified in the SLA, which allows the CSP to support more users and make more profits. A malicious CSP can disrupt the existing SLA monitoring/verification techniques by interfering the monitoring/measurement process. Therefore, we present a SLA verification framework that leverages a third party auditor (TPA). Under the TPA framework, we propose an effective testing algorithm that can detect SLA violations of physical memory size in virtual machine (VM). Using real experiments, we show that the algorithm can detect cloud cheating on VM memory size (i.e., SLA violations). Furthermore, our algorithm can defend various attacks from a malicious CSP, which tries to hide a SLA violation.**

Keywords - Cloud Computing; Service Level Agreement; Verification; Monitoring; Testing

I. Introduction

Cloud computing paradigm is gaining increasing attentions recently because it brings many economic benefits to users. Cloud computing can reduce capital expenditures, such as hardware costs and software license costs, and it also shrinks operational expenditures such as costs of hiring IT personnel significantly. Moreover, users will have universal data access and storage at multiple independent geographical locations [1]. A number of major IT companies (such as Amazon [2], Google [3], IBM [4] and Microsoft [5]) have started offering cloud computing services. Meanwhile, increasing number of enterprises are migrating their tasks to the cloud environment.

Cloud computing is client/mission-oriented, specified by Service Level Agreements (SLAs) between cloud service providers (CSPs) and users in a pay-as-you-go way. A SLA is a contract between the two parties, which states the details of the resources and performances that the cloud should offer. The typical SLA metrics include memory size, CPU speed, storage size, network bandwidth, system uptime, and packet loss. For example, a small instance of the Amazon EC2 has the following configurations [6]: 1.7 GB memory, one 1.0 - 1.2 GHz Opteron or Xeon processor, and 160 GB instance storage.

A SLA serves as the basis for the expected level of services obtained from the CSP. The bill that a user pays to the CSP is closely related with the SLA. How does a user know if he/she is getting physical memory size or CPU speed as specified in the SLA? A CSP is a profit-based company, and it has the incentive to cheat on SLA. For example, a CSP may provide less memory to a user, which allows the CSP to support more users and make more profits.

Obviously, users cannot rely on the CSP to verify the SLA. Now Amazon EC2 puts the burden of verifying SLAs on users. However, it is difficult for a user to verify the SLA because the CSP has complete controls of underlying resources, including physical machines, hypervisors, virtual machines (VMs), et al. In a word, neither the CSP nor the user is suitable for the SLA verification. Hence, a third party should be designed to perform the cloud SLA verification. However, SLA verification in cloud is much more difficult than that in the traditional Internet and computer networks. An untrusted CSP can easily defeat the existing SLA monitoring/verification techniques by interfering the monitoring/measurement process.

In this paper we present a SLA verification framework that leverages a third party auditor (TPA). A TPA resolves the trust dilemma between a CSP and its users. There are several benefits of the TPA-based SLA verification framework. First, the TPA framework is flexible and scalable. It supports various types of tests targeting at different SLA metrics (e.g., memory or CPU). Second, it supports testing from multiple (including a large number of) users, which could significantly enhance the capability of testing a cloud. Third, the TPA framework also relieves users from the verification burden. With the third-party auditing functions, we can either prove to users that the CSP indeed satisfies the SLA (which will in turn build trust between users and the CSP), or detect and report a SLA violation to users (which will protect users' benefit and also deter CSP from cheating in future). The contributions can be summarized as follows.

- We propose a flexible and scalable framework that utilizes a third party auditor for cloud SLA verification. The framework supports various types of SLA tests.
- We design an effective testing algorithm that can detect SLA violations on physical memory size of a VM.
- Using real experiments, we demonstrate that the algorithm can detect cloud cheating on VM memory size (i.e., SLA violation). Our testing algorithm also can defend various attacks from a malicious CSP, which tries to hide

978-1-4673-0921-9/12/$31.00 ©2012 IEEE

a SLA violation.

The rest of this paper is organized as follows. Section II describes the related work on SLAs. Section III discusses our assumptions and the threat model. Section IV presents the TPA framework and Section V discusses the testing algorithm for detecting SLA violations on VM memory size, followed by the conclusion made in this paper in Section VI.

II. Related Work

Studies [7], [8] discuss SLA issues in traditional IP networks. However, as a new paradigm, cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [9]. Work [10] proposes a layered cloud architecture to model the bottom-up propagation of failures, and uses it to detect SLA violations by mapping resource metrics to SLA parameters. There are several approaches for SLA assessment with a focus on accurately measuring or estimating Quality of Service parameters. Study [11] proposes a novel active measurement methodology to monitor whether the characteristics of measured network path are in compliance with performance targets specified in SLAs. Study [12] proposes a new passive traffic analysis method for on-line SLAs assessment, which reduces both the need for measuring QoS metrics as well as the interactions between the ingress and egress nodes in the network. Work [13] presents a quantitative study of the end-to-end networking performance among Amazon EC2 from users' perspective, and concludes that virtualization can cause significant unstable throughput and abnormal delay variations. Study [14] compares the performance and cost of four major cloud providers (Amazon, Microsoft, Google and Rackspace).

However, the previous work did not consider an untrusted cloud that can interfere with the measurement/monitoring process triggered by users. A malicious cloud may intentionally modify, delay, drop, inject or preferentially treat packets in order to disrupt the measurement. Work [15] indeed takes the presence of an adversary into account, but the threat model for an adversary in the middle of a path is different from our work where the adversary (the cloud) is at the end of a path. In addition, they focus on the networking SLAs (such as packet-loss rate and delay). In this paper, we emphasize on a different SLA parameter - the physical memory size of a VM.

III. Assumptions and Threat Model

In this paper, our assumptions are given in the following:

1) Users trust the authorized third party and allow the TPA to perform auditing functions. A user may delegate his account to the TPA for a while to verify the SLA. This assumption is reasonable and sometimes necessary since certain SLA metrics must be measured from the user's VM, such as response time.
2) One thing that can prevent a CSP from cheating is to require the CSP to provide the source codes of its hypervisor to the TPA that can ensure there is no "malicious" code.
3) However, clean static codes do not guaranty a secure running instance because the CSP may run a different version of the hypervisor. Thus, the TPA should be able to examine the integrity of a hypervisor at the run time. This can be achieved by using some existing techniques, such as HyperSentry [16].
4) The CSP allows the TPA to monitor its hypervisor, which ensures that the hypervisor does not perform two tasks: (a) To detect if there is a TPA test running in the cloud; and (b) after detecting a TPA test, the cloud changes the resource allocation of a VM such that the SLA is satisfied. For example, a user leases a VM with 2GB physical memory, but the CSP sets the maximum memory value of the VM to 1GB. When the CSP detects such a test, it switches to 2GB immediately. Then, the TPA will not be able to detect any SLA violation (which actually happened).

Our threat model is given below:

1) The CSP has complete controls of its own resources, including physical machines, hypervisors, VMs, et al.
2) The CSP is able to know any security material (such as an encryption key) used by a VM, because the material is stored in the physical memory and/or the hard drive that the hypervisor has access to. In addition, the CSP is able to modify any message sent by a VM without being detected. For example, if a VM runs a test and creates a timestamp after the test is completed, the cloud can change the timestamp stealthily, even if a message authentication code (MAC) is used. The hypervisor knows the key for the MAC and it can create a new valid MAC after changing the message.
3) The CSP will perform the cheating only if the cost is less than C, where C is a cost-related parameter. For example, if the CSP needs to reserve a GPU such that it is not detected by a GPU/CPU test, the cost of cheating may be too large for the CSP and it has no incentive to cheat.

IV. SLA Verification Framework

Our objective is to develop an auditing solution that can verify whether a CSP satisfies a SLA or not. Given a semi-trusted (or untrusted) CSP, we propose a third party auditor based framework, including three parts as shown in Fig. 1: (1) a third party auditing module (TPAM) running in the cloud machine; (2) the SLA testing programs (testers); and (3) the TPA server that locates outside the cloud.

TPAM is a software module implemented by the TPA to monitor the integrity of a running hypervisor in the cloud. To defend against compromised hypervisor, TPAM's job is to check the consistency between the running instance and the correct executable version. Thus, first, given the hypervisor's source codes, the TPA will examine them by static program analysis to find out malicious operations, and then compile a

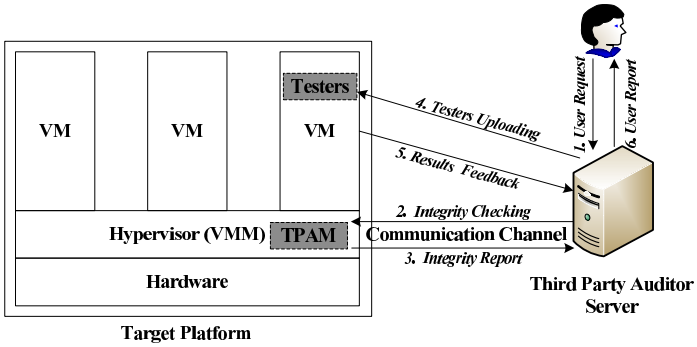

Fig. 1. The SLA verification framework

clean-slate correct executable version with necessary parameters supplied by the cloud. Second, in order to securely test the running hypervisor, TPAM must be trustworthy, which means its execution should not be modified or interrupted by the hypervisor, nor should its results be modified. In this paper we protect the TPAM by using HyperSentry [16], which has the following properties to make it a good candidate to perform our task, i.e., to check if a running hypervisor is the same as the correct executable version:

1) HyperSentry provides a framework to enable an agent to measure the integrity of the highest privileged software (e.g., the hypervisor).
2) HyperSentry can be invoked stealthily without the awareness of the hypervisor.
3) The measurement output can be securely conveyed to a remote verifier. The hypervisor is not able to alter or forge the output.

Under the framework of HyperSentry, the TPAM can be examined step by step by using trusted boot hardware [17] until all components are measured at the boot time. During the trusted boot procedure, the code and data of the TPAM are copied into the SMRAM (a designated and lockable memory) and kept from accessing or modifying, regardless of the process's privilege level. As a result, no software can modify the code and data of the TPAM, and its trust can be maintained.

At the run time the TPAM can be activated by interrupts from the Intelligent Platform Management Interface (IPMI) [18] via an out-of-band channel, which is triggered by the TPA server. IPMI is a server-oriented platform management interface directly implemented in hardware and firmware. Hardware features can be used to differentiate between interrupts generated by the out-of-band channel and other methods. An interrupt generated by the out-of-band channel is the exclusive way to trigger the TPAM.

Finally, the actual checking of the running hypervisor is straightforward. After being activated by the outside TPA server, the TPAM computes a hash of the running hypervisor, securely signs the hash, and then sends the signed hash to the outside TPA server for verification.

SLA Testing Programs are special testing programs (also called testers) proposed in our work, which can defend various attacks from an untrusted cloud while still being able to detect SLA violations. Traditional benchmarks are used to evaluate the performance of certain hardwares in a trusted environment. However, the VMs run in machines owned by the CSP that controls hypervisors and other resources. A malicious CSP may change the measurement result or complete a testing task at a different (more powerful) machine without being detected. Hence, the existing benchmarks cannot be used in our case. Similarly, the existing SLA measurement techniques are not available to verify SLA in an untrusted cloud.

TPA server plays a centric role in the auditing framework and controls the SLA verification process. When the TPA server is ready for a test, it triggers the TPAM, which checks the integrity of the hypervisor. Then the TPA server starts the SLA testing programs remotely. After finishing the tests, it will record and analyze the results to give a proof whether the SLA has been violated.

Specifically, the procedure of our framework has six steps illustrated in Fig. 1:

1) A user initializes a request to the TPA server for a test, such as whether a given resource is satisfied as the SLA claimed. Additionally, the user may delegate his account to the TPA for a while to verify the SLA.
2) Upon receiving the user's request, the TPA server immediately activates the TPAM via the out-of-band channel and makes sure that the hypervisor is clean and correct.
3) The TPAM computes a hash of the running hypervisor, securely signs the hash, and then sends the signed hash to the outside TPA server for verification.
4) The TPA server receives the signed hash and compares it with that of the correct executable version. If the running hypervisor is correct, the TPA server uploads the testers and the corresponding data, and starts a test in the VM.
5) When the test is completed, the TPAM sends the results back to the TPA server.
6) Based on the results, the TPA server determines whether there is a SLA violation and generates a report for the user.

V. SLA Verification on Memory Size

A CSP is very powerful and has complete controls of its resources, including physical machines, hypervisors, VMs, et al. Hence, it is a challenging task to detect SLA violations by an untrusted cloud. In this section, we study one of the most important SLA metrics - memory size, and present an effective algorithm that can detect whether the cloud actually allocates a VM the physical memory size as specified in the SLA.

Note that the usable memory size in a VM should exclude the memory used by the VM and other system softwares. Besides, it is normal for a cloud to schedule physical memory among VMs, because this is the way the CSP benefits. If a user leases a VM (denoted as VM1) with 2GB physical memory, the cloud may allocate less but sufficient memory (may be less than 2GB). However, when the applications

require more memory, the cloud will satisfy VM1 immediately, up to the maximum value (e.g., 2GB in this example). The above behavior is considered normal and the SLA is satisfied.

In contrast, a SLA violation is considered as follows. VM1 is specified with 2GB physical memory in the SLA. When VM1 is running, the hypervisor also tells VM1 (and the user) that its maximum memory is 2GB. However, the hypervisor sets the actual maximum memory of VM1 to 1.5GB, which means that VM1 will never get more than 1.5GB no matter how many processes are running. Thus, when the workload increases, the computations will spend more time than should be in VM1. As a result, all the computations in VM1 suffer from performance degradations.

A. The Access-Time-Based Memory Testing

To detect any SLA violation on VM resources, the key is to design effective testing programs for different resources according to their usage characteristics. It is well-known that the usable memory size is very important for application performance because less usable physical memory will cause more memory page faults and need more swapping operations between the physical memory and the hard disk. And this significantly increases the access/computation time, because the hard drive access time (3 to 5 milliseconds) is much larger than that of the physical memory (2 to 70 nanoseconds) [19]. The access-time-based (ATB) memory testing algorithm is exactly designed based on the above access-time difference.

Denote M as the maximum memory size that is usable by user applications. From our experiments in Amazon EC2, the size of memory used by all the applications in a VM is in fact less than M in most cases. A SLA violation on memory size will not be detected if the memory usage is not close to the value of M. Therefore, the ATB algorithm must satisfy:

- It tries to use a memory size of (or close to) M.
- In order to defeat any cheating from the hypervisor, it must have computations based on actual access (e.g., read) to the physical memory.
- The result should not be predictable by the hypervisor. Otherwise, the hypervisor could pre-compute the result.
- The result should be verifiable by the outside TPA.

As a result, the ATB algorithm is presented below.

1) The TPA server creates an array R of size M.
2) The TPA server sets the value of each element of R (a simple case is $R[i] = i$).
3) The TPA server uploads the array R to VM1, which means that VM1 will create the array R, and set the same value for each element.
4) When the TPA server wants to test the cloud, it generates a random number r, and sends r to VM1.
5) After sending, the TPA server records the time t_1 immediately.
6) VM1 randomly selects N array elements (details given below), and computes the sum of the N elements. This means that VM1 needs to read the N elements from the physical memory. If the cloud does not allocate sufficient physical memory, some of the array elements will not be stored in the physical memory (but rather in the hard disk), which will cause a much longer access time.
7) As soon as the computation is done, VM1 returns the result (i.e., the sum) to the TPA server.
8) The TPA server records the time t_2 when it receives the result.
9) Based on the time t_1 and t_2, the TPA server can figure out the computation time in VM1 (details given below), then the TPA server can determine if the cloud actually allocates sufficient physical memory to VM1.

There are still two issues to be further discussed in the ATB algorithm: the selection of N random elements (step (6)) and how to figure out the computation time in VM1 (step (9)).

In step (6), there are two trivial approaches to generate the random indexes.

Approach #1: The TPA server generates N random indexes and sends them to the tester in the cloud. However, the number of indexes may be large (e.g., could be 50 millions), which may introduce large communication overhead.

Approach #2: The TPA server generates a random seed and sends it to the tester. The tester uses the seed and a pre-stored function to generate N random indexes of array R. However, the hypervisor is able to see the function (stored in physical memory or hard drive) and compute the N random indexes when it sees the random seed. Then the hypervisor can move all the N array elements (corresponding to the N random indexes) into the physical memory before the computation starts, which makes the computation fast enough. Finally, the TPA server will not be able to find out any SLA violation.

In order to select N random elements efficiently, our solution uses a random number r as the index of the first element, and determines the next index based on the value of the current element. In general, the n^{th} index is based on the value of the $(n-1)^{th}$ element $R[n-1]$. Specifically, the n^{th} index is determined by the following equation:

$$n^{th}\ index = \{(n-1)^{th}\ index + \\ lowest\ k-bit\ of\ R[n-1]\}\ mod\ (S)$$

Where k satisfies $2^{k-1} \leq S < 2^k - 1$, and S is the size of the array. The lowest k-bit of the $(n-1)^{th}$ element $R[n-1]$ could be considered as a "random" number between 0 and $2^k - 1$.

In the above method, one has to read the value of the $(n-1)^{th}$ element, in order to generate the n^{th} index. If a malicious CSP wants to cheat, it will be very costly to generate all the indexes using the above method, because the method requires the hypervisor to read all the N elements, which is equivalent to our test. And if some of the N elements are not in the physical memory, it will take much longer time to complete the index generation, which makes any cheating of the hypervisor detectable.

In step (9), the test time $\Delta t = t_2 - t_1$ includes the Round Trip Time (RTT) between the TPA server and VM1, plus the computation time of the test in VM1. However, the RTT may have some variations, which is a noise to the measurement.

TABLE I
COMPARISON OF EXECUTION TIME WITH FULL AND PARTIAL MEMORY ACCESS.

Sample Rate	Δt_{full}(s)	$\Delta t_{partial}$(s)	$\Delta t_{partial}/\Delta t_{full}$
1/10	0.622	7.648	12.296
1/20	0.314	3.824	12.178
1/30	0.213	2.554	11.991
1/40	0.160	1.925	12.031
1/50	0.129	1.552	12.031
1/100	0.068	0.778	11.441
1/200	0.036	0.389	10.806
1/500	0.017	0.156	9.176
1/1000	0.011	0.079	7.182

According to the measurement results [14], the RTT of four major CSPs (Amazon, Google, Microsoft and Rackspace) is less than 200 milliseconds. Therefore, the effect of the RTT noise can be reduced as follows. If the computation time is in the order of several seconds, the variation of the RTT will not affect our decision (i.e., whether there is a SLA violation).

B. Experiments

The real experiments were performed at Harbin Institute of Technology to evaluate the effectiveness of the access-time-based algorithm in a small cloud where XEN [20] was used as the hypervisor. There are two scenarios: (1) the cloud allocates the full memory size as specified in the SLA; and (2) the cloud allocates less memory than that specified in the SLA. The results are reported in Table I, where the sample rate refers to the rate of sampling the array R (for example, a sample rate of 1/10 means the algorithm randomly selects 1/10 elements from the array). The sample rate is used to control the number of elements to be accessed, which in turn controls how large the computation time will be. Δt_{full} is the test time recorded under scenario (1) (i.e., full memory access); and $\Delta t_{partial}$ is the test time recorded under scenario (2) (i.e., with 50% memory access and 50% hard drive access).

Each test was run five times to eliminate the random influences from other applications in the VM. Table I shows the big difference of the execution time between the full-memory access and the partial-memory access. The partial-memory access time is about 7 to 12 times of the full-memory access time. Table I also indicates that the length of the computation time can be adjusted by varying the sample rate. When the sample rate is 1/50, the difference between Δt_{full} and $\Delta t_{partial}$ is larger than one second. Given that RTT is less than 200 milliseconds, all the tests with sample rate higher than 1/50 are able to detect if full physical memory is available (i.e., if SLA is satisfied). The actual RTT in our experiments is less than one millisecond. In general, if there is a big gap between the two times Δt_{full} and $\Delta t_{partial}$, the ATB algorithm is able to verify whether the CSP allocates sufficient memory to the VM.

The effectiveness of the ATB algorithm is also studied when a cloud performs different levels of cheating. The level of cheating refers to the percentage of memory that is not provided to the VM. A small percentage of memory cheating may not be easily detected because it only causes small performance degradation for the user. The results are given in Table II for various percentages of memory cheating where the sample rate is 1/100.

TABLE II
COMPARISON OF EXECUTION TIME FOR DIFFERENT PERCENTAGES OF MEMORY CHEATING (SAMPLE RATE = 1/100).

Percentage of Memory Cheating	Δt_{full}(s)	$\Delta t_{partial}$(s)	$\Delta t_{partial}/\Delta t_{full}$
50%	0.068	0.778	11.441
40%	0.089	0.440	4.944
30%	0.053	0.324	6.113
20%	0.083	0.279	3.361
10%	0.080	0.186	2.325

TABLE III
COMPARISON OF EXECUTION TIME FOR DIFFERENT PERCENTAGES OF MEMORY CHEATING (SAMPLE RATE = 1/10).

Percentage of Memory Cheating	Δt_{full}(s)	$\Delta t_{partial}$(s)	$\Delta t_{partial}/\Delta t_{full}$
50%	0.624	7.683	12.313
40%	0.889	4.387	4.935
30%	0.517	3.693	7.143
20%	0.827	2.765	3.343
10%	0.790	1.853	2.346

Table II gives the execution time for different memory cheating percentages, varying from 50% to 10%. As expected, the smaller the cheating percentage, the less the difference between the two access times. When the cheating percentage is 50%, the partial-memory access time is about 12 times of the full-memory access time. However, when the percentage is 10%, the difference becomes smaller, only 2.33 times or 106 milliseconds.

By using a high sample rate, the ATB algorithm is able to detect even a small percentage of memory cheating. Table III presents the results when the sample rate is 1/10. Even for the 10% memory cheating, the difference between Δt_{full} and $\Delta t_{partial}$ is more than one second, which is large enough to verify a memory cheating (a SLA violation).

C. Security Analysis

For the access-time-based memory testing algorithm, a possible cheating is to migrate the VM to another machine that has sufficient physical memory when a CSP detects such a test. However, according to [21] and [22], even at a fast bandwidth, a VM migration takes 5 - 10 seconds with 0.3 - 1.5 seconds downtime. In contrast, the ATB algorithm takes less than 0.9 seconds to complete (for full memory access). Hence, a VM migration can be detected due to the large migration delay. It is normal that a cloud is allowed to migrate VM when some resource at a machine is running out. However, if a cloud always migrates a VM when it is being tested by the TPA, it is a strong indication of cloud cheating.

A malicious cloud may launch other attacks on the ATB algorithm. For example, if a cloud does not allocate enough memory, it may use only array elements in the physical memory for the computation during a test. However, this attack

can be easily defeated because the wrong computation result will be verified by the outside TPA server that knows the correct result. To sum up, our memory testing algorithm can defeat various cheating/attacks from an untrusted cloud.

VI. Conclusion

In this paper we propose a flexible and scalable framework that leverages a third party auditor for cloud SLA verification. Under the framework, an effective testing algorithm is designed to detect a SLA violation on VM physical memory size. The access-time-based memory testing algorithm utilizes the difference of the access time between physical memory and hard drive. The real experimental results demonstrated that the algorithm can effectively detect SLA violations on VM physical memory size in the cloud. Also, it is shown that our algorithm can defend various attacks from a malicious cloud.

Acknowledgment

This research is supported by the National Basic Research Program (973 Program) of China under Grant No. 2011CB302605, the National High-Tech Development 863 Program of China under grants No. 2011AA010705, 2012AA012506 and 2010AA012504, the National Natural Science Foundation of China under grants No. 60903166 and 61173145, and by the US National Science Foundation (NSF) under grants CNS-0963578, CNS-1022552, and CNS-1065444.

References

[1] Opencrowd cloud taxonomy, [Online]. Available: http: www.opencrowd .com/views/cloud.php

[2] Amazon EC2, [Online]. Available: http://aws.amazon.com/ec2

[3] Google App Engine, [Online]. Available: http: www.google.com/enterp rise/appengine

[4] IBM Cloud, [Online]. Available: http://www.ibm.com/cloud-computing /us/en

[5] Microsoft Azure, [Online]. Available: http://www.microsoft.com/win dowsazure

[6] Amazon EC2 Instance Types, [Online]. Available: http://aws.amazon. com/ec2/instance-types

[7] A. Shaikh and A. Greenberg, "Operations and management of IP networks: what researchers should know," *ACM SIGCOMM Tutorial Session*, 2005.

[8] J. Martin and A. Nilsson, "On service level agreements for IP networks," *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, pp. 855-863, 2002.

[9] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," *National Institute of Standards and Technology*, 2011.

[10] I. Brandic, V. C. Emeakaroha, M. Maurer, S. Dustdar, S. Acs, A. Kertes, and G. Kecskemeti, "LAYSI: A layered approach for SLA-violation propagation in self-manageable cloud infrastructures," *Proceedings of 34th Annual IEEE Computer Software and Applications Conference Workshops*, pp. 366-370, 2010.

[11] J. Sommers, P. Barford, N. Duffield, and A. Ron, "Multi-objective monitoring for SLA compliance," *IEEE/ACM Transactions on Networking*, vol. 18, issue. 2, IEEE Press: NY, USA, pp. 652-665, 2010.

[12] R. Serral-Gracia, M. Yannuzzi, Y. Labit, P. Owezarski, and X. Masip-Bruin, "An efficient and lightweight method for Service Level Agreement assessment," *Computer Networks*, vol. 54, issue. 17, Elsevier: New York, NY, USA, pp. 3144-3158, 2010.

[13] G. Wang and N. T. Eugene, "The impact of virtualization on network performance of Amazon EC2 data center," *Proceedings of the 29th IEEE Conference on Computer Communications*, pp. 1163-1171, 2010.

[14] A. Li, X. Yang, S. Kandula, and M. Zang, "CloudCmp: comparing public cloud providers," *Proceedings of the 10th Internet Measurement Conference*, ACM: New York, NY, USA, pp. 1-14, 2010.

[15] S. Goldburg, D. Xiao, E. Tromer, B. Barak, and J. Rexford, "Path-quality monitoring in the presence of adversaries," *Proceedings of the 2008 ACM SIGMETRICS on Measurement and Modeling of Computer Systems*, ACM: New York, NY, USA, pp. 193-204, 2008.

[16] A. M. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. C. Skalsky, "HyperSentry: enabling stealthy in-context measurement of hypervisor integrity," *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ACM: New York, NY, USA, pp. 38-49, 2010.

[17] Trusted Computing Group, [Online]. Available: http://www.trusted computinggroup.org

[18] IPMI - intelligent platform management interface specification v2.0, [Online]. Available: http://download.intel.com/design0/servers/ipmi/IPMIv2_0rev1_0.pdf

[19] Hard Disk Drive Access Time, [Online]. Available: http://en.wikiped ia.org/wiki/Hard_disk_drive#Access_time

[20] Xen Hypervisor, [Online]. Available: http://www.xen.org

[21] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper, "Predicting the Performance of Virtual Machine Migration," *Proceedings of the 18th Annual Meeting of the IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, IEEE Computer Society: Los Alamitos, CA, USA, pp. 37-46, 2010.

[22] M. Zhao and R. J. Figueiredo, "Experimental study of virtual machine migration in support of reservation of cluster resources," *Proceedings of the 2nd International Workshop on Virtualization Technologies in Distributed Computing*, ACM: New York, NY, USA, pp. 51-58, 2007.