Do you get what you pay for? Using Proof-of-Work Functions to Verify Performance Assertions in the Cloud

Falk Koeppe and Joerg Schneider
Department of Electrical Engineering and Computer Sciences
Technische Universitaet Berlin
Berlin, Germany
{*murphlaw,komm*}*@cs.tu-berlin.de*

***Abstract*—In the Cloud, the operators usually offer resources on a pay per use price model. The client gets access to a newly created virtual machine and has no direct access to the underlying hardware. Therefore, the client cannot verify whether the Cloud operator provides the negotiated amount of resources or only a fraction thereof. Especially, the assigned share of CPU time can be easily forged by the operator. The client could use a normal benchmark to verify the performance of his virtual machine. However, as the Cloud operator owns the underlying infrastructure, the operator could also tamper with the benchmark execution. We identified four attack vectors to modify the results of the benchmark. Based on these attack vectors, we showed that using proof-of-work functions can disable three of them. Proof-of-work functions are challenge response systems, where it is simple to generate a challenge and verify the result while solving the challenge is compute intensive. We implemented three proof-of-work functions in a prototype benchmark. Experiments showed that the runtime of the proof-of-work functions sufficiently relates to the results of the reference benchmark suite SPEC CPU2006.**

***Keywords*-Cloud computing; benchmark; security; trust**

I. Introduction

A classical application service provider is able to sign a service level agreement (SLA) regarding the service performance. Usually, the response time or number of concurrent connections is used as the criterion for service performance. However, when moving from software as a service to platform as a service in the Cloud, the performance properties measurable from the outside depend not only on the service providers' infrastructure. The application installed by the end user and its configuration have a huge impact on the service performance, too. In this scenario, a Cloud operator can hardly guarantee that the common performance metrics measurable from the outside will be hold. Therefore, the client and the Cloud operator have to agree on a metric measuring directly the capacity and speed of the provided infrastructure.

The Cloud operator can verify the performance metrics on the physical computer hosting the virtual machines booked by the client. However, this measurement is not neutral as the Cloud operator can simply lie about the measured value. On the other hand, the client can run a benchmark within its virtual machine only. Measuring computer performance is a widely studied field. But the general assumption is that the machine is in full control of the benchmarker and that the measured hardware does not change before, during, or after the benchmark run. However, the Cloud operator controls the underlying hardware and can easily manipulate the assigned share of resources or exchange data and programs in the virtual machine. We will show that there are a number of ways, how the Cloud operator can tweak the benchmark run to manipulate the result.

Therefore, a Cloud benchmark is needed which is resistant to certain manipulation attacks by the Cloud operator. In this paper, we propose a tamper resistant benchmark based on proof-of-work functions. Proof-of-work functions are special functions that are hard to solve and while at the same time the result can be verified without much effort.

First, we discuss the properties of benchmark programs to measure computer performance and present related work in this area. Then, we introduce proof-of-work functions and their key properties. Based on the properties needed for a Cloud benchmark, we select three well known proof-of-work functions. In the following section, we discuss the attack vectors to manipulate the benchmark result and show how a proof-of-work based benchmark can disable three of four. Before concluding, we compare the measured values of our prototype implementation with a reference benchmark.

II. Benchmarking Computer Performance

Benchmark programs assess the capacity and speed of computer systems. In our scenario we will focus on the speed factor, as the capacity, e.g., available memory, hard disk, number of special components, is not that easily forgeable in the Cloud as the processing speed. In the speed factor, we will focus on the general processing speed, i.e., how fast can the computer solve a given problem or how many problems can the computer solve in a given time.

A. Benchmark Programs

There exist a wide variety of different benchmark programs that span industry-standard benchmark suites to simple tools for performance analysis. To get a better overview on what benchmarks do and how they achieve the task of measuring computer performance, we will describe some of the most popular and well-known benchmarking programs.

As we will see, different benchmarks often focus on different subsystems of the computer system or differ substantially because of other factors. One benchmark might focus more on measuring the processor, others analyze the performance of the I/O subsystem and again others incorporate a mix of applications to create a scenario that resembles more closely the everyday usage of the computer system. Benchmarks also differ in the algorithms or applications they use, even though they all focus on the processor for example. Further differences lie in the size or length of the benchmarks: Some test only few scenarios and finish within minutes while others need several hours to complete to simulate many application scenarios. You have to keep in mind that the development and choice of applications is often subjective in the sense that certain companies and stakeholders influence the decision of what is part of the benchmark and what is not. Kevin Castor points out the pitfalls an analyst can encounter by interpreting a benchmark result in [1].

The following state of the art benchmarks are designed to assess the processing speed and, therefore, could fit in our scenario to benchmark Cloud system:

LMbench: [1] is a small and simple benchmarking program that is focused on measuring the latency and bandwidth of a computer system [2]. The benchmark determines the time needed for certain processes, context switching, communication latencies, system latencies, and the communication bandwidths in MB per seconds. LMBench is quite old and current test results are not updated on the webpage any longer but it is still commonly used, e.g., in the current Debian and Ubuntu Linux distributions.

Linpack Benchmark: [2] is a benchmark that clearly focuses on CPU performance of supercomputers. It solves a series of linear equations and therefore measures the floating-point computer performance. This benchmark is used to rank high performance computer systems in the TOP500 list[3]. The result of the benchmark is represented as millions of floating-point operations executed per second (MFLOPS/s). Despite the advantages over the similar measurement units, David L. Lilja points out in [3] that the unit of MFLOPS/s is unreliable, inconsistent, and not independent. Matthias Bolten also passes criticism on the Linpack Benchmark in [4] and provides alternatives that would overcome the problems.

SPEC CPU2006: [4] is an industry-standardized benchmark that stresses the processor, the memory subsystem and the compiler of a computer system. SPEC was founded 1988 to provide realistic measurement data of the performance of different computer systems, instead of relying on the data given by the marketing departments of the computer vendors. The benchmark uses a wide mix of real world applications to measure the computer performance under the workload of a real usage scenario. Because of the importance and magnitude of the benchmark, vendors tend to optimize their systems and compilers for the various SPEC tests.

The presented benchmarks are only a subset of the numerous available programs. However, they are a good sample of the existing approaches.

Benchmarking the Cloud can have multiple meanings. The term is sometimes used for testing a benchmark using Cloud resources, for benchmarking software without the need of own hardware, and for assessing real hardware before buying it. As discussed before, our focus is the performance evaluation of the virtual machine booked in the Cloud to detect cheating Cloud operators.

Binnig et al. discussed the specific requirements and objectives for Cloud benchmarks [5]. They explored possible ways to benchmark many of the new features in the Cloud using new application scenarios. However, the authors do not address the uncertainties due to cheating Cloud operators.

To monitor the performance of booked virtual machines, Cloud operator provide monitoring tools like CloudWatch[5] by Amazon. It shows the CPU usage, the network I/O and the disk I/O and allows the automatic provision of load balancer and auto scaling. Obviously, the tool is not independent from the Cloud operator. Therefore, it cannot be used to neutrally asses the performance. However, our tamper resistant benchmark can be used to verify the results of CloudWatch.

III. Proof-of-Work Functions

In this section we are going to introduce the properties of proof-of-work functions and identify the key properties for a Cloud benchmark function. Three proof-of-work functions are then selected for our prototype implementation.

Proof-of-work functions were developed to deter denial of service attacks or service abuses such as spam. The key feature of proof-of-work functions is the dissymmetry between the amount of work needed to compute the function and the one needed to verify the correctness of the result. Therefore, it can be used in a challenge-response system. The challenger easily generates a challenge and verifies the result while the challenged partner needs to perform complex computations. Usually, the result of the challenge will be thrown away after verifying the correctness. It is still an open question whether it is possible to create proof-of-work function producing usable output. In general, the difficulty of the challenge can be adjusted, i.e., the expected average amount of resources needed can be set.

[1] http://www.bitmover.com/lmbench/
[2] http://www.top500.org/project/linpack/
[3] http://www.top500.org/
[4] http://www.spec.org/cpu2006/
[5] http://aws.amazon.com/cloudwatch/

Dwork and Naor first mentioned proof-of-work functions in [6], where they propose to use a computational challenge to fight spam. By now, a lot of different proof-of-work functions were developed, which partly focus on different attributes and features or have special use cases in mind.

Proof-of-work functions are successfully used in the fields of deterring denial of service attacks [7], connection depletion and service abuses such as spam [8], but are also proposed to measure the popularity of a web page [9], to fairly handle overload situations [10], to serve as a currency for peer-to-peer and grid applications [11], and to realize the micropayment schemes PayWord and MicroMint [12].

In the following, we discuss the different properties of proof-of-work functions and how they relate to their use as a Cloud benchmark.

Cost: An ideal proof-of-work function would require always the same resources. However, most functions do not guarantee *fixed costs*. The *bounded probabilistic* functions can at least guarantee an upper bound for the time to solve a challenge, while for *unbound probabilistic functions* only the average resource usage is specified.

Stability: For the probabilistic functions a low variance in the runtime is favorable. Some authors propose to repeat the proof-of-work function multiple times and use the average runtime to reduce the variance [13], [10].

Security: There should be no shortcut available to compute the proof-of-work result with fewer resources. To realize this property, most functions are based on NP hard problems or other mathematical hard problems.

Concurrency: Most proof-of-work functions are based on a walk through a huge search space. Usually, such functions can be easily parallelized resulting in much lower runtimes. Such a parallelization is no violation of the security property defined above as at least the same amount of resources is used.

Advantage: The advantage of a proof-of-work function was first introduced in [13], where it is defined as the ratio between the number of steps needed to check the validity of the result and the number of steps needed to solve the proof-of-work function. Generally, we are interested in an algorithm with a high advantage.

Parameterability: This attribute describes the ability to change the difficulty of the challenge to be solved. Especially, the granularity to fine tune the expected amount of resources needed is important. For many functions, the problem space can be doubled only which leads to an exponential increase in the run time of the function.

CPU-bound vs. memory-bound: Some proof-of-work functions are restricted by the CPU speed while others use a lot of memory to stress the memory bandwidth. The memory-bound functions are based on the fact that memory bandwidth does not evolve as fast as CPU speed. Furthermore, within one computer generation it is more likely that the memory bandwidth is comparable even if the CPU speed varies. Thus, memory-bound functions are considered more balanced when different clients have to proof their interest. However, for the Cloud benchmark, we are specifically interested in the differences in the performance. Hence, we consider only CPU-bound functions.

We identified four of these attributes to be essential for proof-of-work benchmark functions used within a cloud environment:

- As discussed before, the function has to be *CPU-bound*.
- In order to get representative results of the benchmark, we need a *high stability* (low variance). Otherwise, each run of the benchmark would lead to another measured value. In this case, the Cloud operator could argue that the computer was fast enough but the specific challenge was too hard to be solved in the expected time.
- As we want to test a wide range of virtual machines in a reasonable short time, the benchmark has to adapt to the processing speed. Therefore, we need a *high parameterability* to fine tune the expected amount of work.
- The client books resources in the Cloud in order to save costs for own hardware. Also, verifying the benchmark results should require no special hardware. Therefore, we need a proof-of-work function with a *great advantage*.

There exist a number of proof-of-work functions that often have specialized attributes or were developed with a certain application in mind. Not all functions can be used in the context of measuring the computer performance. The Table I gives an overview on a number of CPU-bound functions and categorizes them according to the remaining three main attributes.

Dwork and Naor defined three proof-of-work-functions based on mathematical hard problems [6]. The first function is based on the mathematical problem to find the square root of an arbitrary x in the congruence classes modulo p where p is prime. Here the advantage is very high as finding the square root requires at least $\log p$ steps and verifying the result is a simple multiplication. The other two are based on the Fiat-Shamir and Ong-Schnorr-Shamir signature schemas. The idea is to forge a cryptographic signature. The first schema by Fiat and Shamir is weakened to make it breakable. For the second schema by Ong, Schnorr, and Shamir, the Pollard algorithm is used to break it. The first schema needs 2^k steps to solve the challenge (with k tuning the difficulty) and k multiplications to verify the result while the second needs only three multiplications for the verification and $\log N$ steps to find a solution (with N being the product of two primes also acting as the difficulty parameter). The last two approaches benefit from the discussions on the security of the underlying signing schemas. However, they are rather complex to implement especially to implement them in a cryptographically sound

Proof-of-work function	**Stability**	**Parameterability**	**Advantage**
Extracting square root [6]	+/-	+/-	+
Weaken Fiat-Shamir signatures [6]	+/-	+/-	+
Ong-Schnorr-Shamir signature [6]	+/-	+/-	+
HashCash [14]	-	+/-	+/-
HashCashLin [10]	-	+	+/-

Table I
THERE IS NO PROOF-OF-WORK FUNCTION PROVIDING OPTIMAL VALUES FOR ALL BENCHMARK ORIENTED PROPERTIES. HOWEVER, THESE CANDIDATES CAN BE USED TO ASSESS THE PERFORMANCE OF VIRTUAL MACHINES IN THE CLOUD.

way.

Instead of breaking the whole cryptographic signature, another class of proof-of-work functions looks for collisions in cryptographic hash functions. The HashCash function calculates a string such that the SHA-1 hash value starts with a given number of zero bits [14]. Verifying the result requires only one application of the hash function while finding a string is only possible with brute force. The number of zero bits acts as the parameter to steer the difficulty. However, this tuning is rather rough as it only reduces the search space by half. Therefore, the general principle was extended in HashCashLin to enhance its parameterability [10]. Both algorithms are very probabilistic: In some lucky cases the solution is found in the first step; while in other cases many candidates have to be tested. The author of HashCashLin also proposes to run multiple measurements to lower the variance. His experiments showed that already few repetitions reduce the variance to an acceptable level.

For our experiments, we used HashCash, HashCashLin, and extracting square root to see how proof-of-work functions can act as benchmark functions.

IV. BENCHMARKING THE CLOUD

The general question is: How can we make sure that the Cloud operator cannot tamper the benchmark execution? We assume the Cloud to be an untrusted environment as the only reason to execute the benchmark is the lack of trust in the Cloud operator.

The basic assumption is that the Cloud operator can detect the execution of the benchmark and can interfere with its execution from the outside of the client's virtual machine. The Cloud operator has full access to the hardware and the hypervisor software hosting the virtual machines. Nearly all accesses to the virtualized hardware from within the client's machine end up in the Cloud operator hypervisor before they are forwarded to the physical hardware. While modern architectures try to reduce the number of hypervisor interactions, the Cloud operator could decide to disable such features. From the hypervisor, the Cloud operator has also full access to the memory of the virtual machine and can read and manipulate loaded programs and the client's data.

In the following we are going to discuss a number of attack vectors for the Cloud operator based on the given assumption.

Benchmark Skip: In order to increase the predictability, some benchmarks calculate the same formula with the same input values. So the generated output is the same, too. Thus, the Cloud operator could exchange the benchmark program with a fake program which waits a given time and prints the expected result.

Pre-Calculation: If the result of the benchmark is not fixed, but depends only on limited number of input configurations; the Cloud operator could precalculate all possible results and employ the same strategy as above.

Code and Data Exchange: If the benchmark does not use the best known algorithm to solve the problem, the Cloud Operator could exchange the benchmark program with a faster implementation. Sometimes, it is also possible to simplify the problem in a way that the results look similar to the expected. For example, the Cloud operator could reduce the details of a 3D scene to be rendered by the benchmark. A normal user would hardly see the differences in the resulting images.

Additional Resources: The Cloud operator adds more resources to the user's virtual resource during the benchmark run. After finishing the benchmark, the user would get again fewer resources. The Cloud operator could also use additional, even specialized hardware to solve the challenges of the benchmark.

A classical benchmark can be attacked using all these four attack vectors. Using a proof-of-work function as the base of the benchmark disables the first three attack vectors: The result always depends in a non-trivial way on the challenge send by the Cloud user. If the domain of the challenges is large enough, a precalculation of the results for each possible challenge gets too demanding. As the result can be easily verified by the user, the problem cannot be simplified. Additionally, one of the key properties of Proof-of-work functions was the security, i.e., that no shortcut is known to reduce the complexity.

The last attack vector cannot be addressed from within the user's virtual machine. The proof-of-work based benchmark can be tricked by using additional resources. As the client can run the benchmark at any time, the Cloud provider has to add more resources very quickly. However, providing additional resources on very short notice is only possible if sufficient free resources are available. Thus, the costs to

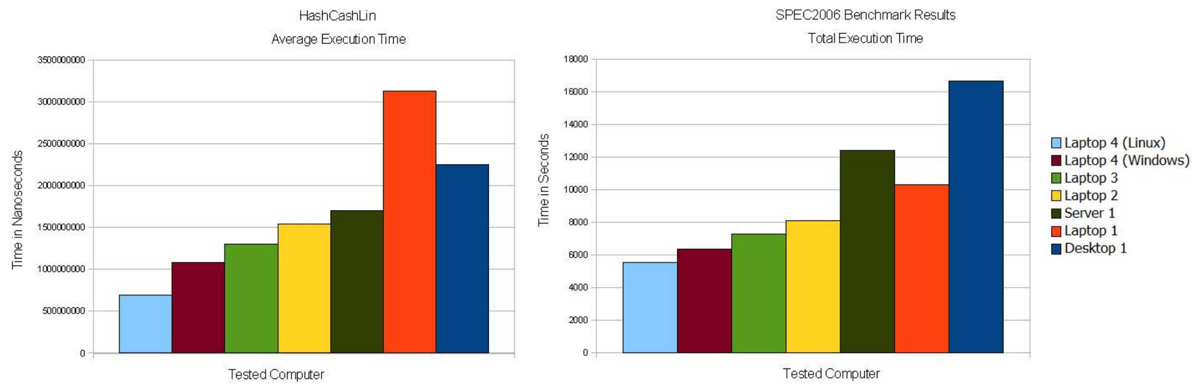

Figure 1. There is a general correlation between the runtime of the proposed benchmark (here based on the HashCashLin proof-of-work function) and the reference benchmark SPEC CPU2006.

perform such an attack are not much lower than to provide the agreed resources.

V. Evaluation

To evaluate the properties of the proposed benchmarking schema, we implemented a prototype in Java using three different proof-of-work functions. The results of the prototype are compared with the results of the SPEC CPU2006 reference benchmark. Furthermore, we looked at the proof-of-work specific properties like variance.

To see how the proof-of-work based benchmark behaves on a wide range of computer we used a sample of six computers of different types and generations:

- *Desktop 1* AMD Athlon XP 2500+ with 2 GB RAM and Windows XP desktop computer
- *Laptop 1* Intel Pentium M Processor 1.86 GHz with 1 GB RAM and Windows Vista laptop computer
- *Server 1* IBM System x3755 2.00 GHz with 4 GB RAM and Windows Server 2003
- *Laptop 2* Intel Core2 CPU T7200 2.0 GHz with 2 GB RAM and Windows Vista laptop computer
- *Laptop 3* Intel Core2 CPU T7600 2.33 GHz with 2 GB RAM and Windows XP laptop computer
- *Laptop 4* Intel Core2 Duo CPU T9300 2.5 GHz with 4 GB RAM and Windows Vista and Linux (Ubuntu 8.04) laptop computer

On each computer, we executed the SPEC CPU2006 benchmark and our proof-of-work based prototype with each proof-of-work function.

In figure 1 we compare the runtimes of the SPEC CPU2006 benchmark with the HashCashLin benchmark. The scale is different, as the SPEC benchmark with its various tests requires more than an hour. On the other hand, the proof-of-work based benchmark is more a micro benchmark concentrating on a single performance value. However, the results show a general correlation between the SPEC CPU2006 values and the proof-of-work functions. But it also makes it clear that the short proof-of-work

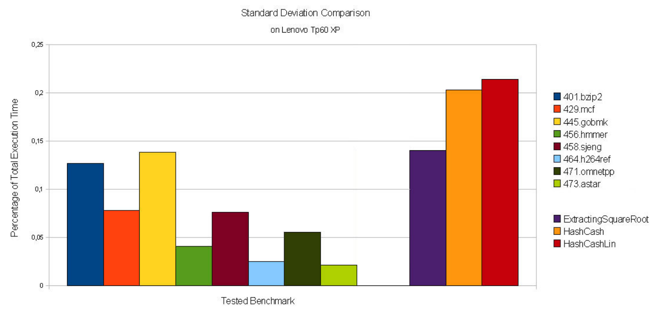

Figure 2. The variance in the runtime of the proposed benchmarks is higher but only by a factor of two.

benchmark cannot be as precise as the full SPEC CPU2006 benchmark suite. The other two proof-of-work functions behaved similar.

Another problem is the probabilistic nature of the proof-of-work benchmarks. Figure 2 shows the standard deviation of some of the SPEC benchmarks and of the three proof-of-work benchmarks. The SPEC benchmarks are also not free of variance. However, due to its probabilistic nature, the proof-of-work benchmark has a higher variance. Nonetheless, the experiments showed that our proposed benchmarks have only less than twice the SPEC variance. Therefore, the results are repeatable and representative for the available computing speed.

VI. Conclusion and Outlook

We introduced a novel tamper resistant benchmark approach to assess the performance of instances in the Cloud. Our benchmark is based on proof-of-work functions and enables the Cloud client to verify the compute performance within its virtual machine. After discussing classical benchmark programs, we discussed the proof-of-work functions and identified the key properties a proof-of-work function needs to be used as a benchmark in the Cloud. We selected three functions and implemented them in a prototype benchmark. Experiments showed that the accuracy of this prototype was comparable with the reference benchmark SPEC CPU2006. Furthermore, we discussed possible manipulations by the Cloud operator and identified four attack vectors. One of them cannot be addressed from within the virtual machine. However, we showed that our novel benchmark disables the other three attack vectors.

In future work, we plan to extend the prototype to periodically measure the capacity without much impact on the actual service provided. Furthermore, we plan to integrate the benchmark into the application to reduce the threat of adding more compute capacity for the benchmark only.

REFERENCES

[1] K. Castor, "Hardware Testing and Benchmarking Methodology," Online; accessed 9-November-2009, February 2006, http://www.donutey.com/hardwaretesting.php. [Online]. Available: http://www.donutey.com/hardwaretesting.php

[2] L. McVoy and C. Staelin, "lmbench: portable tools for performance analysis," in *ATEC '96: Proceedings of the 1996 annual conference on USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 1996, pp. 23–23.

[3] D. L. Lilja, *Measuring Computer Performance: A practitioner's guide*. cambridge university press, 2004.

[4] M. Bolten, "Supercomputer-benchmarks," Online; accessed 19-November-2009, July 2006, http://www.tecchannel.de/server/extra/443198/supercomputer_benchmarks/. [Online]. Available: http://www.tecchannel.de/server/extra/443198/supercomputer_benchmarks/

[5] C. Binnig, D. Kossmann, T. Kraska, and S. Loesing, "How is the Weather tomorrow? Towards a Benchmark for the Cloud," *Proceedings of the 2nd International Workshop on Testing Database Systems*, pp. 1–6, June 2009.

[6] C. Dwork and M. Naor, "Pricing via Processing or Combatting Junk Mail," *Lecture Notes In Computer Science*, vol. 740, pp. 139–147, 1993.

[7] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten, "New Client Puzzle Outsourcing Techniques for DoS Resistance," in *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2004, pp. 246–256.

[8] A. Juels and J. Brainard, "Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks," *NDSS*, p. 15, 1999.

[9] M. K. Franklin and D. Malkhi, *Auditable Metering with Lightweight Security*. London, UK: Springer-Verlag, 1997.

[10] S. Golze, "Fairness in berlastsituationen mittels Proof-Of-Work Funktionen," Ph.D. dissertation, Technische Universitt Berlin, Juni 2009.

[11] F. Garcia and J. Hoepman, "Off-line karma: A decentralized currency for peer-to-peer and grid applications," *Lecture Notes in Computer Science*, vol. 3531, pp. 364–377, 2005.

[12] R. L. Rivest, "PayWord and MicroMint: two simple micropayment schemes," in *CryptoBytes*, 1996, pp. 69–87.

[13] J.-y. Cai, R. Lipton, R. Sedgewick, and A.-C. Yao, "Towards uncheatable benchmarks," *Structure in Complexity Theory Conference, 1993., Proceedings of the Eighth Annual*, May 1993.

[14] A. Back, "Hashcash - A Denial of Service Counter-Measure," August 2002.