Software Execution Protection in the Cloud

[Extended Abstract]

Miguel Correia
INESC-ID, IST, UTL
Lisbon, Portugal
miguel.p.correia@ist.utl.pt

ABSTRACT

Most cloud computing services execute software on behalf of their users. Many war stories and several studies suggest that such software execution is threatened by accidental arbitrary faults and malicious insiders. We present two lines of work to protect software execution in the cloud. The first replicates tasks to protect MapReduce executions from the effects of accidental arbitrary faults. The second uses hardware-based security to protect software execution from malicious insiders.

Categories and Subject Descriptors

D.4.5 [**Operating Systems**]: Reliability–Fault-tolerance; C.2.0 [**Computer-Communication Networks**]: General–Security and protection; C.2.4 [**Distributed Systems**]: Distributed applications

General Terms

Cloud computing, MapReduce, Malicious insiders

1. MOTIVATION

Many companies and organizations are moving their systems to the cloud. A recent report states that the adoption of cloud computing has the potential to generate over 763 billion euros of cumulative economic benefits over 2010-2015, just in the five largest European economies[1]. However, both accidental faults and security events keep plaguing cloud offerings and possibly slowing down cloud adoption.

Many subsystems of the cloud are designed to tolerate both hardware/software crashes and the corruption of data stored in disks, which are very common problems [2]. However, other *accidental arbitrary faults* that can affect the correctness of the results of software executing in the cloud are known to happen. A recent study of DRAM errors in a large number of servers in Google datacenters for 2.5 years concluded that these errors are more prevalent than previously believed, with more than 8% DIMMs affected by errors yearly, even if protected by error correcting codes (ECC) [8]. A Microsoft study of 1 million consumer PCs shown that CPU and core chipset faults are also frequent [5].

The *malicious insider* is for long known to be a difficult security problem [4]. Two recent stories show that the problem can happen in the cloud. A Google engineer was fired early 2010 after the company discovered that he has read Gmail messages and even contacted a group of teens that used Gtalk. Another critical event happened in a cloud storage company called CyberLynk when an ex-employee accessed the company systems and deleted a whole season of a TV series from which there was no other copy (March'11).

2. ACCIDENTAL ARBITRARY FAULTS

MapReduce is a framework developed by Google for processing large data sets [3]. Google's implementation is not openly available, but an open source version called Hadoop is used by many cloud computing companies [9]. Some cloud providers like Amazon (AWS) and Microsoft (Windows Azure) are also providing MapReduce as a service. MapReduce consists basically in executing first a large set of map tasks, then a typically small number of reduce tasks. In Hadoop a job execution is controlled by the JobTracker and tasks are executed by TaskTrackers. In Hadoop input and output data is stored in the HDFS filesystem.

Hadoop (and Google's) MapReduce mostly tolerates crashes of map and reduce tasks. If a task stops abnormally, a timeout expires and a new instance of the same task is created. Additionally, data is stored in disk together with checksums, which allow its corruption to be detected. On the contrary, it does not tolerate accidental arbitrary (or Byzantine) faults, which can affect the correctness of its results.

A simplistic solution to make MapReduce tolerate arbitrary faults would be the following. Consider that f is the maximum number of replicas of the same task that fail. First, the JobTracker would start $2f + 1$ replicas of each map task in different servers and TaskTrackers. Second, the JobTracker would start also $2f+1$ replicas of each reduce task. Each reduce task would fetch the output from all map replicas, pick the most voted results, process them and store its output in HDFS. In the end, the client would pick the most voted output. An even more simplistic solution would be to run a consensus, or Byzantine agreement between each set of map

[1] http://uk.emc.com/microsites/2011/cloud-dividend/

EWDCC'12, May 8, Sibiu, Romania

task replicas and reduce task replicas. This would involve even more replicas (typically $3f + 1$) and more messages exchanged.

In a recent paper we proposed an efficient arbitrary fault-tolerant Hadoop MapReduce [1]. The first above-mentioned simplistic solution is expensive because it replicates everything $2f + 1$ times: task execution, map task inputs reading, communication of map task outputs, and storage of reduce task outputs. We used a set of techniques to avoid these costs: *Deferred execution:* arbitrary faults (excluding crashes) are uncommon, so the JobTracker starts only $f+1$ replicas of the same task; the reduce tasks check if they all return the same result; if a timeout elapses or results do not match, more replicas are started. *Tentative reduce execution:* waiting for $f+1$ map results introduces delay; to avoid it, the JobTracker starts executing the reduce tasks just after receiving the first copies of the required map outputs; in parallel the matching of the inputs is validated; in the unlikely case of not matching, the reduce task is restarted. *Digest outputs:* sending map outputs can be expensive, so reduces pick only one output and f hashes for each map. *Tight storage replication:* reduce tasks are already replicated so they write their results in HDFS only once, instead of using the typical HDFS replication factor of 3.

An experimental evaluation shows that with $f = 1$ the execution of a job with our algorithms uses twice the resources of the original Hadoop, instead of the 3 or 4 times more that would be achieved with the direct application of common Byzantine fault-tolerance paradigms. We believe this cost is acceptable for critical applications that require that level of fault tolerance.

3. MALICIOUS INSIDERS

Clouds based on the Infrastructure as a Service (IaaS) model provide users virtual machines (VMs), storage or networking. Here we are interested in execution, thus on VMs, in which cloud users can execute web application servers and other software. The cloud provider has a large set of servers that run a hypervisor, on top of which user VMs are executed. Recently we have shown that an administrator can easily extract credentials and other confidential data from cloud user VMs [7]. The problem is that he can access the administration VM that runs on servers and obtain a snapshot of the user VM memory, or access its files.

We proposed a solution to protect the execution of software in VMs, as well as the data it manipulates [6]. User VMs reside in the cloud in three places: in servers, in the network during deployment and migration, and on disk. To protect their VMs, users keep their VMs encrypted on the network and disk, and only provide the decryption key to a servers they can trust. We call these servers a trusted virtualization environment (TVE). A TVE comprises a hypervisor and a management VM that do not provide certain operations to administrators (such as snapshots and volume mount) and support only trusted versions of others (launch, migrate, and backup VMs).

Users have to obtain reliable information that a server is a TVE. This can be done using a measurement of the software stack in the server (boot loader, hypervisor, management VM). A measurement is a vector with hashes of these components. To trust it, it has to be provided by a hardware component isolated from the rest of the machine. The Trusted Platform Module (TPM) is a hardware chip that can be used with this purpose. During the boot process each of software module (boot loader, etc.) stores a hash of the next module in a PCR register in the TPM. When the user wants to assess if it can trust a server, it runs an attestation process: the server sends to the user a vector of PCRs/hashes signed by its TPM; the user verifies the signature and if the hashes correspond to a trusted configuration. The user sends the server the decryption key or not depending on the outcome of the attestation.

4. ACKNOWLEDGMENTS

This work was done jointly with Alysson Bessani, Pedro Costa, Marcelo Pasin, Salvador Abreu, Francisco Rocha. It was partially supported by the EC through the TCLOUDS project, and by the FCT through projects RC-CLOUDS, CloudFIT, and FTH-Grid, the Multiannual Program, and the INESC-ID Multiannual PIDDAC Program.

5. REFERENCES

[1] P. Costa, M. Pasin, A. Bessani, and M. Correia. Byzantine fault-tolerant MapReduce: Faults are not just crashes. In *Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science*, pages 32–39, 2011.

[2] J. Dean. Large-scale distributed systems at Google: Current systems and future directions. Keynote speech at the 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware (LADIS), Oct. 2009.

[3] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation*, pages 137–150, Dec. 2004.

[4] M. Hanley, T. Dean, W. Schroeder, M. Houy, R. F. Trzeciak, and J. Montelibano. An analysis of technical observations in insider theft of intellectual property cases. Technical Note CMU/SEI-2011-TN-006, Carnegie Mellon Software Engineering Institute, Feb. 2011.

[5] E. B. Nightingale, J. R. Douceur, and V. Orgovan. Cycles, cells and platters: an empirical analysisof hardware failures on a million consumer PCs. In *Proceedings of the 6th ACM SIGOPS/EuroSys European Systems Conference*, pages 343–356, 2011.

[6] F. Rocha, S. Abreu, and M. Correia. The final frontier: Confidentiality and privacy in the cloud. *IEEE Computer*, 44(9):44–50, Sept. 2011.

[7] F. Rocha and M. Correia. Lucy in the sky without diamonds: Stealing confidential data in the cloud. In *Proceedings of the 1st International Workshop on Dependability of Clouds, Data Centers and Virtual Computing Environments*, 2011.

[8] B. Schroeder, E. Pinheiro, and W.-D. Weber. DRAM errors in the wild: a large-scale field study. In *Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems*, pages 193–204, 2009.

[9] T. White. *Hadoop: The Definitive Guide*. O'Reilly, 2009.