Towards a Self-Adaptive Data Management System for Cloud Environments

Alexandra Carpen-Amarie

Towards a Self-Adaptive Data Management System for Cloud Environments

Alexandra Carpen-Amarie*
INRIA/IRISA, Rennes, France
Email: alexandra.carpen-amarie@inria.fr

I. Introduction

Recently, important academic and industrial actors, such as Google, Amazon or Yahoo!, have started to investigate *Cloud computing*, an emerging paradigm for managing computing resources. Various Cloud computing infrastructure platforms have been proposed as alternatives for clusters or grids, which are the typical infrastructures for large-scale distributed applications. This kind of applications are usually critical in various fields like high-energy physics, climate modeling or bio-informatics. As data volumes generated and processed by these applications increase, a key requirement that directly impacts the adoption rate of the Cloud paradigm is efficient and reliable data management.

More specifically, storage systems intended for very large scales have to address a series of challenges, such as a scalable architecture, data location transparency or high throughput under concurrent accesses. Although these requirements are the prerequisites for any efficient data-management system, they also imply a high degree of complexity in the configuration and tuning of the system, with possible repercussions on the system's availability and reliability. Such challenges can be overcome if the system is outfitted with a set of self-management mechanisms that enable an autonomic behavior, which can shift the burden of understanding and managing the system's state from the human administrator to an automatic decision-making engine. However, self-adaptation is impossible without a deep and specific knowledge of the state of both the system and the infrastructure where the system is running. It heavily relies on introspection mechanisms, which play the crucial role of exposing the system's behavior accurately and in real time.

This PhD research focuses on enhancing a distributed data-management system with self-management capabilities, so that it can meet the requirements of the Cloud storage services in terms of data availability, reliability and security.

II. Related work

In the context of Cloud environments, data availability and fault tolerance are key issues in the design of storage services, as they have to provide competitive service level agreements to the customers. The most representative storage service for infrastructure-oriented Clouds is Amazon's Simple Storage Service (S3) [1], a highly-scalable and reliable web-service that allows users to store data in Amazon's data centers. While the implementation of Amazon S3 is proprietary, its interface has become the "de facto" standard for such storage platforms. Open-source Cloud environments, such as Nimbus [2] or Eucalyptus [3], have designed specific storage services, which are interface-compatible with S3, but also share its requirements, namely strong guaranties for data availability and security. These issues can be overcome by encapsulating a set of self-management mechanisms into the systems, so as to enable them to handle the dynamism and complexity of large data sets. Moreover, even though these systems provide access control list (ACLs) for the users, no high-level security mechanism is available to protect the environment from complex attacks that cannot be prevented by authentication mechanisms.

Besides specific storage systems for Clouds, paradigms such as MapReduce [4] have emerged to meet the needs of data-intensive applications. The MapReduce programming model is based on two functions specified by the user: *map* parses key /value pairs and passes them as input to the *reduce* function. The framework takes care of splitting the data, scheduling the tasks and executing them in parallel on multiple machines. In this context, specialized distributed file systems have been proposed to deal with specific access patterns that require support for highly concurrent and fine-grained access to data. Hadoop [5] is an open-source MapReduce implementation that introduced HDFS [6], while Google relies on Google File System (GFS) [7] for their MapReduce implementation. Whereas GFS provides fault tolerance by constant monitoring and automatic replication of data, HDFS has only limited support for data re-replication and load balancing among storage nodes. However, both systems lack complex self-adaptation mechanisms that require comprehensive information related to data distribution and access patterns to optimize the system behavior on the fly.

III. Our approach

The purpose of this thesis is to build an autonomic, efficient and secure storage service for Cloud environments by leveraging BlobSeer, a large-scale distributed data-sharing platform.

* Advisers: Gabriel Antoniu, INRIA/IRISA, Rennes, France, gabriel.antoniu@inria.fr; Luc Bougé, ENS Cachan/Brittany, IRISA, Rennes, France, Luc.Bouge@bretagne.ens-cachan.fr

A. BlobSeer

BlobSeer [8] is a data-sharing system that addresses the problem of efficiently storing massive data in large-scale distributed environments. It deals with large, unstructured data blocks called BLOBs that are split into equally-sized *chunks*. The architecture of BlobSeer is based on five actors. The *data providers* store the data chunks in a distributed manner, thus enhancing the support for a large number of concurrent operations. Each BLOB chunk is associated with some metadata, which are stored on distributed *metadata providers*. The *provider manager* keeps track of the existing data providers and implements the allocation strategies that map new chunks to available data providers. The *version manager* deals with the serialization of the concurrent requests and publishes a new BLOB version for each write operation. The BlobSeer *client* exposes an interface to user applications. It implements client-side operations for each type of interaction with the BlobSeer system: create BLOBs, read a range of chunks from a BLOB, write or append data to a BLOB.

Extensive performance evaluations carried out for BlobSeer show that it is able to handle massive unstructured data, while providing an efficient support for heavily-concurrent data accesses, thus becoming a potential candidate for a Cloud storage service. To fully fit the requirements of a standalone storage service within a Cloud infrastructure, BlobSeer's behavior can be optimized through a set of self-management features that address important Cloud storage vulnerabilities, such as elasticity, data-availability or security.

B. Introspection mechanisms for BlobSeer

In order to provide support for a self-adaptive behavior of the BlobSeer system, we enhanced BlobSeer with introspection capabilities. In [9], we proposed a three-layered architecture on top of BlobSeer, which generates and analyses BlobSeer-specific data obtained by monitoring the activity of each of its actors. It is designed to identify and generate relevant information related to the state and the behavior of the system, which can be fed as input to various higher-level self-* components. These data are yielded by an (1) *introspection* layer, which processes the raw data collected by a (2) *monitoring* layer. The lowest layer is represented by the (3) *instrumentation* code that enables BlobSeer to send monitoring data to the upper layers.

The instrumentation layer: enables BlobSeer components to generate and send information related to the events that the BlobSeer nodes respond to.

The monitoring layer: has to handle the non-trivial task of gathering data coming from all the instrumented BlobSeer nodes and to make them available to the upper layer that can interpret them. For the implementation of this layer, we relied on the MonALISA [10] monitoring system, as it is a reliable and extensible system that allowed us to define specific data to be monitored and stored.

The introspection layer: processes the data received from the monitoring layer, by implementing a set of data filters at the level of the monitoring services to aggregate the BlobSeer-specific data. We designed a flexible storage schema for the monitored parameters, which pass through the data filters and then are sent to a set of distributed storage servers. We also built a caching mechanism for the storage servers, so as to enable them to cope with bursts of monitoring data generated when the system is under heavy load.

C. A generic framework for enforcing security policies

In order to provide a high-level security mechanism for Cloud storage systems, we proposed a generic framework for both security policies definition and enforcement [11].

The core of this framework is represented by a *Policy Management* module, where security policies definition and enforcement take place. This module is designed to be completely independent of the Cloud system, as its input only consists in user activity events monitored from the system. To access user events, it relies on the *User Activity History* module, a container for monitoring data collected through monitoring mechanisms specific to each storage system. In order to provide security mechanisms specifically tuned for BlobSeer, we used the introspection architecture to generate the *User Activity History*, as it is able to generate all the relevant data needed by the security framework. However, the security framework is designed to be generic, so that it can be employed in conjunction with any system that can monitor and store relevant user activity.

The *Policy Management* framework has a modular architecture, comprising three components:

- The *Policy Definition* component provides a generic and easily extensible framework for defining various types of security policies, which describe inappropriate or dangerous client behavior.
- The *Policy Enforcement* component is responsible for making a decision based on the state of the system and on the impact of the attempted attack on the typical performance of the system. Such decisions range from preventing the user from further accessing the system to logging the illegal usage into the activity history.
- The *Security Violation Detection Engine* scans the *User Activity History* in order to find the malicious behavior patterns defined by the security policies. When such an attack is detected, the *Policy enforcement* component is notified and a set of possible feedback actions are forwarded to it.

IV. Obtained results

We performed a series of experiments to evaluate the introspection layer and to assess the performance of the self-

protection capabilities in BlobSeer. The experiments were conducted on the Grid'5000 [12] testbed, a large-scale experimental Grid platform, that covers 9 sites geographically distributed across France.

A. Visualization tool for BlobSeer-specific data

We implemented a visualization tool that can provide a graphical representation of the most important parameters yielded by the introspection layer. We processed the raw data collected by the monitoring layer and extracted the higher-level data within the introspection layer, by aggregating the parameters collected from BlobSeer. The visualization tool provides synthetic images of the most relevant events in BlobSeer, such as the evolution of the physical parameters (e.g., CPU load, memory), the storage space on each provider and at the system level, the BLOB access patterns or the distribution of the BLOBs across providers.

B. Impact of the introspection architecture on the Blobseer data-access performance

This experiment is designed to evaluate the impact of using the BlobSeer system in conjunction with the introspection architecture. We analyzed the throughput of the BlobSeer operations obtained when deploying the system without any monitoring mechanism and compared it to BlobSeer outfitted with the introspection layers. Since the introspective layer computes its output based on the monitored data generated for each written chunk, the more fine-grained BLOBs we use, the more monitoring information has to be processed. For this test, we deployed 150 data providers and a number of clients ranging from 5 to 80, each of them writing 1 GB of data to BlobSeer. The obtained results show that the performance of the BlobSeer operations is not influenced by the introspection architecture, the intrusiveness of the instrumentation layer being minimal even when the number of generated monitoring parameters reaches $10,000$, as it is the case when testing with more than 80 clients.

C. Evaluation of the security framework

To validate our security framework we needed to see how it performs in large scale Cloud environments. We integrated it into BlobSeer, as it has to be able to handle malicious attacks and to isolate users that initialize them in order to address the security requirements of a standalone storage service within a Cloud infrastructure.

We evaluated the impact of enforcing security policies on top of the BlobSeer system and the performance of the policy management module through a series of large-scale experiments, with a deployment setting consisting of 70 BlobSeer nodes, 8 monitoring services and up to 50 concurrent clients.

Common access patterns for Cloud storage services imply write-intensive and read-intensive scenarios, which have an inherent security vulnerability: Denial of Service (DoS) attacks. To assess the impact of concurrent DoS attacks on the performance of the storage system, we performed three sets of experiments:

- We evaluated the evolution in time of the average throughput of concurrent clients that write to BlobSeer when the system is subject to DoS attacks. The results show that the initial average throughput has a sudden decrease (up to 70%) when the malicious clients start attacking the system. As the *Policy Management* module detects the policy violations, it feeds back this information to BlobSeer, enabling it to block the malicious clients, so that the throughput of the remaining clients increases back towards its initial value.
- We measured the impact of concurrent DoS attacks on the performance of the storage system. When all the concurrent writers act as correct clients, the system is able to maintain a constant average throughput for each client, around 110 MB/s. However, when no security mechanism is employed, the performance is drastically lowered while several clients attempt an attack, decreasing under 50 MB/s when more than 30 clients are deployed, out of which 50% are malicious. Further, the throughput increases again, once the attackers are blocked by the security framework.
- In order to efficiently protect BlobSeer against security threats, the *Policy Management* module has to expose attacks as fast as possible, so as to limit the damage inflicted to the system and to minimize the influence on the correct clients. We measured the detection delay when the percentage of malicious clients increases from 10% to 70% out of a total of 50 concurrent clients, showing that the system is able able to promptly react when an attack is initiated. The first malicious client is detected in 20 seconds and the last one is detected in about 55 seconds, while the duration of the write operation increases towards 40 seconds when 70% of clients perform a DoS attack.

V. Development directions

To introduce a self-adaptive behavior in BlobSeer, we are investigating several directions, which match the main self-management properties defined for autonomic systems [13]:

Self-configuration through dynamic data providers deployment: This is a means to support storage elasticity in BlobSeer, by enabling the data providers to scale up and down depending on the system's needs in terms of storage space and access load. We designed a component that adapts the storage system to the environment by contracting and expanding the pool of data providers based on the system's load.

Self-optimization through automatic data replication and removal strategies: Since many data-intensive applications require high data availability, a data-management system has to automatically maintain the replication degree

of data chunks and to support a dynamic adjustment of the replication degree, according to the load of the storage nodes and the applications access patterns. Furthermore, the clients can benefit from configurable data removal strategies, allowing the system to automatically delete data that is seldom accessed or is temporary generated by the applications.

Self-protection through security policies enforcement: We aim to develop our security framework by introducing a *Trust management* module, which will dynamically compute a trust value for each user based on his past actions and on the real-time system state. The trust values will enable the system to support adaptive security policies specifically tuned for the history of each user.

Another direction we investigate is to integrate BlobSeer with an existing Cloud infrastructure, such as the Nimbus Cloud environment. Our goal is to expose BlobSeer as a Cloud storage service compatible with existing Cloud storage interfaces. To this end, we interfaced BlobSeer with Cumulus, the storage management component in Nimbus, designed to be interface-compatible with Amazon S3. Preliminary results show that the BlobSeer storage back end is able to sustain a promising data transfer rate, while bringing an efficient support for concurrent accesses.

VI. Conclusions

The emergence of Cloud computing brings forward many challenges that may limit the adoption rate of the Cloud paradigm. As data volumes processed by applications running on Clouds increase, the need for efficient and secure data management emerges as a crucial requirement. This work aims to enable BlobSeer, a large-scale data-management system, as a Cloud data service, by addressing a series of self-management requirements.

The first step towards an autonomic behavior was to equip the BlobSeer platform with introspection capabilities, which can serve as input data for a self-adaptive engine designed to address such goals as self-configuration, self-optimization or self-protection. We developed the self-protection direction within a generic security management framework allowing providers of Cloud data management systems to define and enforce complex security policies. In addition, we designed an expressive policy description language enabling system administrators to define a large array of security attacks and to enforce various types of restrictions upon the detected malicious clients.

Finally, we are integrating our data-management system as a storage back end within the Nimbus Cloud environment.

References

[1] Amazon Simple Storage Service (S3). http://aws.amazon.com/s3/.

[2] K. Keahey, R. Figueiredo, J. Fortes *et al.*, "Science Clouds: Early experiences in cloud computing for scientific applications," *In Cloud Computing and Its Application 2008 (CCA -08) Chicago*, 2008.

[3] D. Nurmi, R. Wolski, C. Grzegorczyk *et al.*, "The Eucalyptus open-source cloud-computing system," in *Proc. 9th IEEE/ACM Int. Symposium on Cluster Computing and the Grid*, Los Alamitos, USA, 2009, pp. 124–131.

[4] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[5] D. Borthakur, *The Hadoop Distributed File System: Architecture and Design*, The Apache Software Foundation, 2007.

[6] "HDFS. the Hadoop distributed file system," http://hadoop.apache.org/common/docs/r0.20.1/hdfs_design.html.

[7] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," *SIGOPS - Operating Systems Review*, vol. 37, no. 5, pp. 29–43, 2003.

[8] B. Nicolae, G. Antoniu, L. Bougé *et al.*, "BlobSeer: Next generation data management for large scale infrastructures," *J. Parallel Distrib. Comput.*, Aug 2010.

[9] A. Carpen-Amarie, J. Cai, A. Costan *et al.*, "Bringing introspection into the BlobSeer data-management system using the MonALISA distributed monitoring framework," in *First Int. Workshop on Autonomic Dist. Syst. (ADiS 2010)*, Krakow, Poland, 2010, pp. 508–513.

[10] I. Legrand, H. Newman, R. Voicu *et al.*, "MonALISA: An agent based, dynamic service system to monitor, control and optimize grid based applications," in *Computing for High Energy Physics*, Interlaken, Switzerland, 2004.

[11] C. Basescu, A. Carpen-Amarie, C. Leordeanu *et al.*, "Managing data access on clouds: A generic framework for enforcing security policies," in *The 25th International Conference on Advanced Information Networking and Applications (AINA-2011)*, Singapore, 2011.

[12] Y. Jégou, S. Lantéri, J. Leduc *et al.*, "Grid'5000: a large scale and highly reconfigurable experimental grid testbed." *Intl. Journal of High Performance Comp. Applications*, vol. 20, no. 4, pp. 481–494, 2006.

[13] A. Carpen-Amarie, A. Costan, J. Cai *et al.*, "Bringing introspection into BlobSeer: Towards a self-adaptive distributed data management system," *Int. Journal of Applied Math. & Computer Science*, vol. 21, no. 2, To appear in 2011.