Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Environmental Sciences 10 (2011) 691 – 696

Procedia Environmental Sciences

2011 3rd International Conference on Environmental Science and Information Application Technology (ESIAT 2011)

Building a Cloud Storage Service System

Chengzhang Peng[a], Zejun Jiang[b], a*

[a,b] *School of Computer Science and Technology, Northwestern Polytechnical University, Xi'an 710072, China*

Abstract

Cloud Storage services are increasingly noticed as they promise elastic capability and high reliability at low cost. In such services, you can store most of your files to authenticated Cloud Storage Service center, and you do not worry about your space being inadequate or wasted because the storage being able to be adjusted dynamically is the most important feature of the Cloud Storage. In this paper, we present a solution about how to build a Cloud Storage Service System based on the open-source distributed database, it follows a stratum design that includes Web service front-end, transformation processing layer and data storing layer. Terminal users can access their own data in this system through three Web service interfaces. More over, a complete prototype system based on this architecture is demonstrated.

Keywords: Cloud Storage Service; Amazone's S3; Distributed Database; SOAP; REST

1. Introduction

Cloud computing regards infrastructure, platform, and software as services, which are made available as orderbased services in a pay-as-you-go model to users. In industry, these services are respectively referred to as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). The importance of these services is highlighted in a recent report from Berkeley as: "Cloud computing, the long-held dream of computing as a utility, has the potential to transform a large part of the IT industry, making software even more attractive as a service" [2]. According to the current researching achievements about cloud computing, several research fields such as web computing, grid computing, distributing computing and virtual computing highly contributed to today's cloud computing.

* Corresponding author. Tel.: +8613379081097; fax: +8602988857916.
E-mail address: abelard2009@mail.nwpu.edu.cn.

doi:10.1016/j.proenv.2011.09.111

It is sure that cloud computing must go with cloud storage, that is why cloud storage are increasingly noticed. Although sometimes cloud storage's success is considered not technology-driven but economical, various research and commercial cloud storage systems presented in Tim's thesis [6] told us there are many interesting areas.

Cloud storage service is an emerging infrastructure that offers Platforms as a Service (PaaS). In industry, Amazon Simple Storage Service (Amazon S3) is regarded as the best reference of cloud storage service. Amazon S3 provides virtually elastic and unlimited storage space, we can access the internet-based storage service through using a set of simple interfaces. Users pay for the service of the amount of storage, Get Request, Put Request, data transfer in and data transfer out each month for one year.

If reading Amazon S3 API [1], we will summarize several features of Amazon S3 as follows: First, users create buckets that can contain the arbitrary objects up to 5 terabytes in size, each accompanied by up to 2 kilobytes of meta-data, generally, and we refer to buckets as containers and objects as basic unit of user data. Second, users upload or download own entire objects, each up to 5G in one operation, and objects are identified within each bucket by a unique, user-assigned key through REST and SOAP interfaces. To access objects, HTTP is the primary protocol. Third, accessing Amazon S3 buckets or objects' requests use a customized HTTP scheme based on a Hash Message Authentication CODE (HMAC) [7]. We will get both Access Key ID and Secret Access Key when registering an account, and therefore, every request at least includes your Access Key ID, selected elements and its signature which is the output of using your Secret Access Key to calculate the HMAC of selected elements. Upon receiving an authenticated request, Amazon S3 service gets a Secret Access Key that you claim to have, and then compute a "signature" for the message it received in the same way. It then compares the signature it calculated with the signature presented by the requester. If both signatures match, the system thinks that the requester must have access to the Secret Access Key, and so that has the authority of the principal to whom the key was issued. The request is given up and the system responds with an error message if both do not match. Fourth, Amazon S3 holds eventual consistency, therefore, users may not get their own newest content of an object identifier very soon after it has been overwritten.

In open-source, Eucalyptus [9] is an infrastructure for cloud computing. It is built in the Amazon EC2, in which users configure virtual machines in the cloud and run tasks on them. Eucalyptus mainly includes four components. The bottommost level component is node controller, One Eucalyptus may have one or more nodes, a node controller executes on every node. The node controller queries and controls the host operating system and the hypervisor on its node. The middle-level component is cluster controller, One Eucalyptus system may have several clusters, every cluster needs a cluster controller to schedule resource in its own cluster and controller network connecting to both the nodes running node controller and the machine running. The two top-level components are node controller and storage controller (Walrus); the former offers a web interface for cloud management and EC2-compatible SOAP, and performs high-level resource scheduling and system accounting; the latter offers S3-compatible cloud storage service.

In cloud storage system Dynamo, a highly available key/value store in cluster environment developed by Amazon, chooses to provide "always writable" data availability with the trade-off eventual consistency. Cassandra [8] is a distributed storage system for managing very large amounts of structured data spread out across many commodity servers. Cassandra provides highly available service with no single point of failure with the use of peer-to-peer model.

One attention we must pay is that cloud storage system is different from cloud storage service system. In general, the former can not provide service to the popular users because these systems have not easy-to-use API (as the manual of Amazon's S3 describes).

Although Amazone's S3 is the most successful product, there is not much discussion about the implementation of it. At the same time, the large-scale distributed data storage once again becomes hot

research project, but the discussion about how a distributed data storage can become a cloud storage service system analogy to Amazon's S3 is still absent.

In this paper, we present a solution about how to build a cloud storage service system based on the open-source distributed database, terminal users can access their own data in this system in several manner with which they are familiar; And this system will support the maximum number of concurrent connections; at last we present our prototype of cloud storage server system based on Cassandra.

The remainder of this paper is organized as follows. In section 2, we present a proposed three-tiered architecture of cloud storage service system-CS3 and discuss every system components and our prototype system based on the architecture in detail. Section 3 summarizes this paper and our future work.

2. System Design

In this section, we propose a general architecture of a cloud storage service system-CS3. The architecture of the CS3 is simple, flexible and modular with a hierarchical design reflecting common resource environments found in many academic settings. By and large, the system allows users to access their own data using SOAP and REST interfaces. Of course, for the system's better compatibility, we will implement an emulation of Amazon S3's SOAP and REST interfaces in the near future. In addition to the programmatic interfaces, CS3 also offers a Web interface for users. Using a Web browser, users can upload, download, and look through their own data in CS3, which is similar to on-line disk you used.

To offer the service of the above described CS3, Figure 1 depicts the proposed architecture of CS3. This system consists of three main stratums: a Data Storage layer, a Transformation Processing layer, and a Web system layer. The key challenge is how to design techniques for each layer component and make these components work together in a coherent system.

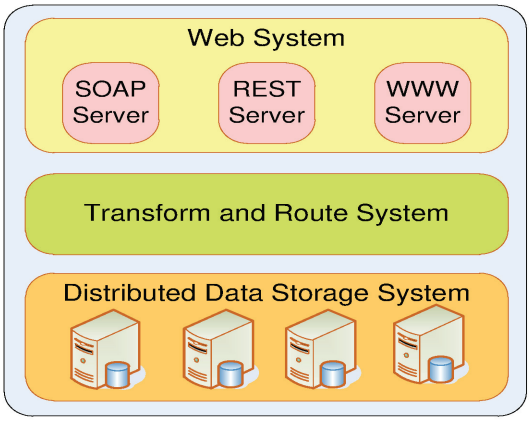

Fig. 1. Stratum architecture of a cloud storage service

In storage layer, the distributed database (e.g., Cassandra [8]) distributed file system (e.g., HDFS [4]) or parallel file system (e.g., PVFS [5]) were successfully used in some famous projects. Whatever scheme is chosen, the final goal is that the storage system must be highly scalable and highly available. In our project, we use distributed database as the storage component. Transformation Processing Layer mainly changes different type of Web requests (e.g., SOAP or REST) into uniform bottom-level data storage request. As more and more users want to share the cloud storage service, several different interfaces must be provided, so that three primary types of Web service interfaces in Web System Layer will be discussed.

We will explain every component in CS3 in detail, and discuss the relationship between them.

2.1. Data Storing Layer

According to the rules of pay-by-use model, and cloud computing services being organized as a utility, our data storing layer must be able to meet following requirements, the most important feature is dynamic scalability being achieved whenever storage nodes could be easily added into or removed from the system; the second feature is consistency, and so forth. From Point of providing one kind of cloud storage service(e.g., Amazon S3 mainly being used to store videos, photos, audio), we can choose one in Cassandra, CouchDB, HBase, Redis, Scalaris, Project-Voldemort, and so on.

One key point is that we need provide the same interface to uses, but our implementations have many differences because these databases have different architectures, data models and client APIs. We take Cassandra and HBase for instances as follows.

HBase: HBase [3] is a distributed, columned-oriented store modeled after Google's Bigtable and is a Hadoop database, which is an open-source MapReduce framework. Just as Bigtable, HBase relies on a distributed file system and lock service, and provides Bigtable-like ability on top of Hadoop. The combination of the file system and Hadoop is called HDFS [6].

From HBase's data model [3], its applications need store in labeled tables' row. All data rows always are arranged in row key's lexicographic order, and a data row can hold an arbitrary number of columns. Every column name is form "<family>: <label>" where <family> and <label> can be arbitrary byte arrays.

Cassandra: Cassandra is a decentralized, structured, distributed key-value store system. Every node in the Cassandra's cluster is identical. There are no network bottlenecks and no single points of failure. It ingeniously brings together the decentralized technologies from Dynamo and the data model from Google's BigTable, therefore from view of the concept, a table in Cassandra may be thought of a collection of rows that are located by a row key and where any column may not have a value for a particular row key. One thing worthy to be pointed out is that Cassandra provides a column families-based data model richer than typical key/value store systems, Cassandra implements two kinds of columns families, Simple and Super column families. Super column families can be viewed as a column family within a column family.

2.2. Transformation Processing Layer

In this layer, the received SOAP, REST, or Web-based request located in the front server such as Tomcat will be transformed into the above described database request. One major challenge for implementing this layer is that user should can choose his/her favorite one of popular programming languages, and one of three kinds of internet interfaces to connect to the front server, but the database request only can be generally implemented in one programming language (e.g., Java or Erlang), and the getting/putting data request generally also should be relatively single. Of course all these mainly belong to physical implementation details, in theory, this layer can be shown as Figure 2, it includes five components. We will detailedly describe every component as follows:

WS Request Web Service component analysis every request from Web server (e.g., Tomcat), validate the request parameters, at the same time, validate whether the request is synchronic or asynchronous.

WS Response According to the response type, we will encapsulate the data from Data Structuring component in the corresponding response.

The input of data structuring component comes from the analysing of WS Request or DB Response components. If it comes from the former, we will structure the request parameter into the distributed

database client request; if from the latter, we will structure the response parameters into Web service responses based on the different service request type.

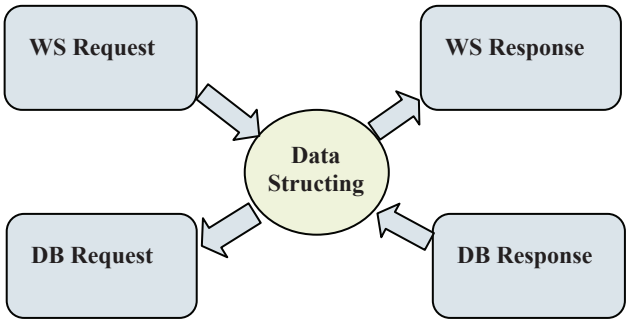

Fig. 2. The model of the transformation processing

DB Request/Response In this component, we will encapsulate the data from Data structuring component into the distributed database client request, or send the response from the database to the Data Structuring component.

2.3. Web System Layer

In general, this layer is located in the front server with Tomcat or other Web application server software. It includes SOAP, REST and Web-based server software, which are usually deployed in Tomcat or other Web application server software. When the Web application server receives a request, it will activate the corresponding server (e.g., SOAP server) to process this request. Implementing an available Web System being able to response SOAP, REST and Web-based request is not difficult thing, because there are many open-source components such as Axis, Tomcat and so on.

2.4. Prototype System

We build our prototype system CS3 according to the above proposed architecture. CS3 contains 3 Cassandra nodes, which are located on 3 independent FC12 installed on VMware whose host machine is HP ProLiantML150 G6 Server, a Web server with Apache Tomcat equipped with Intel(R) Core(TM)2 Duo CPU E7500 @ 2.93GHz processor, 3 GB of memory, 300GB SATA disks.

We used Cassandra as the distributed database layer. Then, we develop a transformation processing module, it can transform a SOAP request into Cassandra client request, and transform a Cassandra response into SOAP response parameters. In Web system layer, we deploy SOAP services on the Tomcat server. According to data model features of the Cassandra, we design our own data model as shown in Figure 3.

We use WSDL to describe SOAP service similar to Amazon's S3. In this prototype system we define three major APIs for this service system:

get (filename) get one file from CS3 according to the provided file's name.

put (fileAttributes, fileContents) upload a file and its attributes.

delete (filename) delete a file according to the provided file's name.

Of course, we can use Amazon's WSDL file for compatible with S3 if necessary.

In addition, we develop a terminal user application with SWT for validating our cloud storage service system.

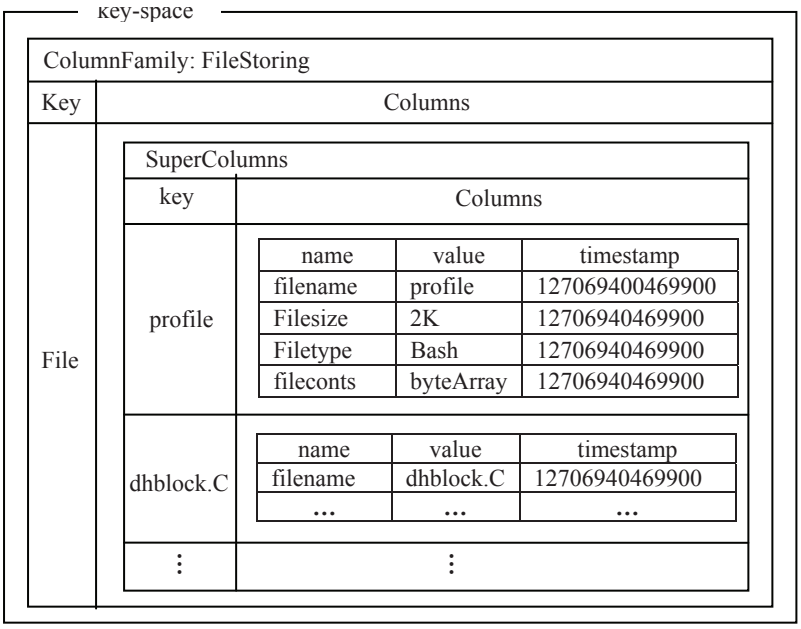

Fig. 3. Data model of prototype system

3. Conclusion

In this paper, we provide a solution for building a CS3 based on the open-source tools. Our proposed CS3 is designed as three-tiered architecture. According to this architecture, every interested researching team all can build own CS3 for better understanding cloud storage service (e.g., Amazon's S3) which has a great deal of hype. We demonstrate our prototype of CS3 built based on the framework in this paper and detailedly described component.

Of course, there are many things to do for us. In the near future, we will add security module and heavily concurrent processing module which are necessary to a complete CS3.

4. References

[1] Amazon simple storage service api reference, api version. *Amazon Web Services LLC*; 2006.

[2] Armbrust M, Fox A, Griffith R, Joseph AD, Katz RH, Konwinski A, Lee G, Patterson DA, Rabkin A, and Zaharia M. Above the clouds: A berkeley view of cloud computing. *Technical report*; 2009.

[3] Chapter 10: Architecture. *The Apache Hbase Book*. Apache Software Foundation; 2010.

[4] The Apache Software Foundation. Hdfs architecture. *http://hadoop.apache.org/hdfs/docs/current/hdfs_design.html*; 2009.

[5] Haddad IF. Pvfs: A parallel virtual file system for linux clusters. *Linux J.*; 2000.

[6] Kraska Tim. Building Database Applications in the Cloud. *PhD thesis*; 2010.

[7] Krawczyk H, Bellare M, Canetti R. Hmac: Keyed-hashing for message authentication; 1997.

[8] Lakshman A, Malik P. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*; 2010,44:35–40.

[9] Nurmi D, Wolski R, Grzegorczyk C, Obertelli G, Soman S, Youseff L, Zagorodnov D. The eucalyptus open-source cloud-computing system. *IEEE International Symposium on Cluster Computing and the Grid*; 2009, p. 124–131.