Distributed System, Assignment3
Yanran Du, a1701762
Ziyang Ye, a1707805

Topic : App Engeine

1. Introduction

Google App Engine (GAE) belongs to the PaaS layer (Platform as a Service) in the cloud computing system. It mainly provides users with a limited free "runtime environment" cloud computing platform, allowing users to "register" the application system. "On Google's servers and take advantage of the cloud computing capabilities provided by the Google infrastructure.

By using GAE, developers can run and maintain applications on Google's servers, and applications can easily scale to grow based on traffic and data storage needs.

2. Architecture

The Google app engine is a web application service platform,and its architecture can be divided into three layers:

The first layer, data storage. Relying on the rebuilt Google infrastructure such as “GFS” and “BigTable”, it provides a environment of application service that can handle high-load and large-capacity data scales with high performance and high scalability. There are its main features: dynamic network services, full support for common network technologies; persistent storage with queries, classifications and transactions; automatic extension and quantization balance features; API of user authentication.

The second layer, web server. Under the CGI standard protocol, Google appengine application holds a communication with the server. The Web server accepts a “Get” or a “Post” request from the client, and the application processes the output data and submits it to the Web server for presentation on the client. Unlike old-school web hosting environments, Google app engine does not provide files directly from the application's source directory, but you can modify the engine to use static files. For security reasons, Google app engine builds a secure sandbox environment that isolates network application services in a secure space, which means it is independent of any hardware architecture, operating system, and real-world web server location.

The third layer, development layer. With a code base and modules, it supports a variety of programming languages. In addition, Google app engine allows you to upload any other third-party libraries to develop and construct your web application.

3. Heterogeneity

3.1 Definition

The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks. Heterogeneity applies to hardware device, operating system and programming language.

3.2 Protocol Buffers

Google app engine use Protocol Buffers technology to solve heterogeneity in services: application servers are connected to many services, and heterogeneity problems may occur. For example,application servers are written in Java, and some services are written in C++. Google's solution in this regard is based on language-neutral, platform-neutral and extensible Protocol Buffer, and all API calls through the Google app engine must be compiled into Protocols before RPC (Remote Procedure Call). Buffer's binary format.

Protocol Buffer, a language-neutral, platform-neutral, and scalable way to serialize structured data. It supports various programming languages like Java, C++, and Python. Each of these languages support both compiler and library files in binary, so it is 10 times faster than exchanging data with XML. The Protocol Buffer is used in two aspects: 1) RPC communication: The communication between distributed applications; the communication in heterogeneous environments. 2) Data storage: Protocol Buffer has self-documenting and easy to compress features, so it can be used to persist data, like storing log cat. And it can be processed by the MapReduce.

3.3 MapReduce

As we mentioned MapReduce, so there is going to be a brief introduction about this kind of distributed computing model. First, Google data center receives a large amount of data that need to be processed. Since many of these data are large, usually reaching at the PB level, therefore, parallel processing is required. The MapReduce programming model is introduced to solve this problem. MapReduce is derived from a functional language, and it distributes the processing of massive data to each node separately, thus realizing the enforceability and fault tolerance of the work. It mainly uses two steps "Map" and "Reduce" to process a large-scale data set in parallel. Any of the “Map” functions in the figure below is an operation on the input (Original Data). They do not have affinity with each other, and they are all done in a parallel environment. Any “Reduce” operation is a merge operation. It partially merges the results generated by the “Map” operation. Each Reduce operation is also independent. The final result is the complete result set required for the entire work.

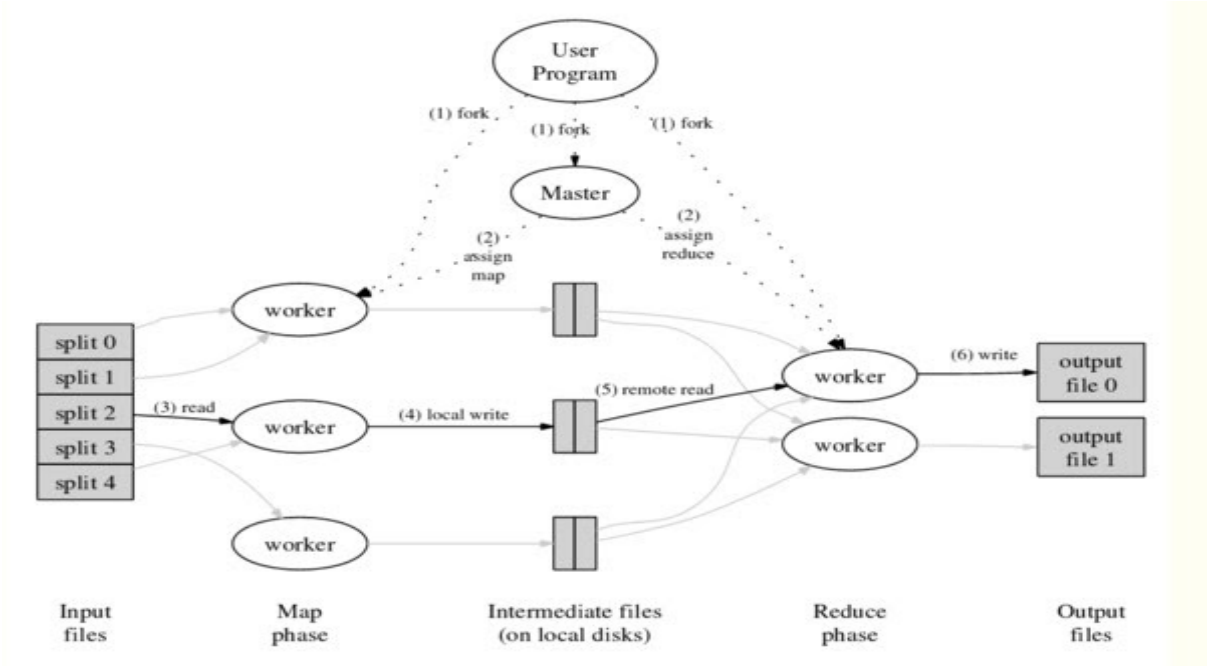

Figure 1. MapReduce Opertaion Mechanism

4. Scalability

4.1 Introduction

A system is scalable if it can handle an increase in users and resources without causing significant performance loss or increased management complexity. Without doubut, the Google app engine is very scalable. It relies on one of the core technologies of google cloud computing: google file system "GFS".

4.2 GFS

GFS is an extensible distributed file system, and it is used for a large, distributed and massive data access application. It runs on inexpensive, regular hardware but provides fault tolerance. It can provide a large number of users with a higher overall performance service. The use of the central service model reduces the design difficulty, determines the non-cached data according to the necessary nature and feasibility, and implements the file system in the user mode, so that the user can more easily utilize a variety of debugging tools, and Operating systems run in different spaces and are more independent of each other, facilitating separate upgrades of their own systems and kernel systems.

The GFS system architecture is shown in the figure2. The whole system is divided into three roles: Client, Master and Chunk Server. Client is the access interface of the application, and the application can directly call the function in the Client library. Master is the main server, responsible for the entire system, and is the core part of the entire file system. Chunk Server is primarily responsible for storing data in the form of files.

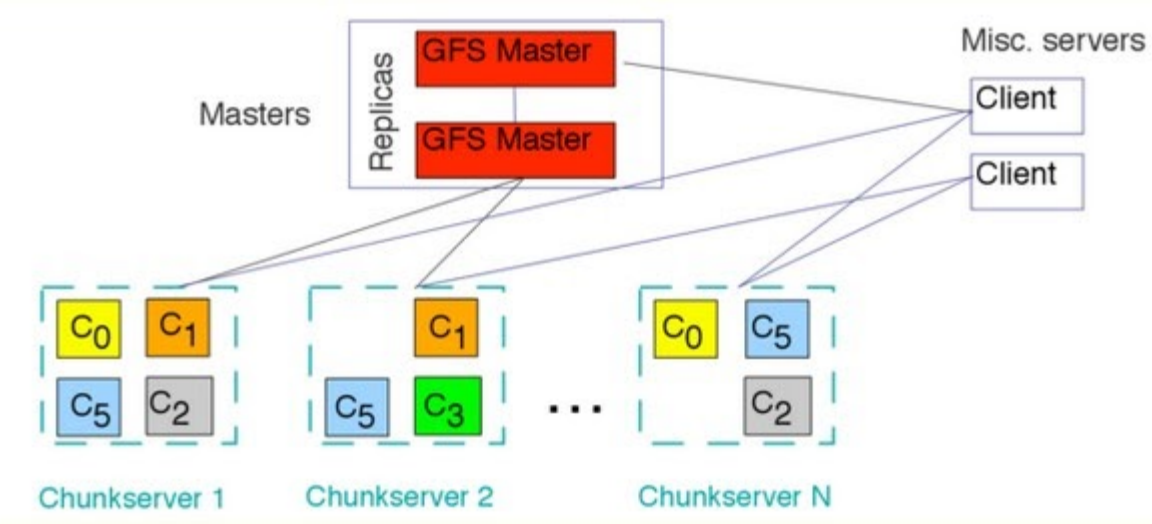

Figure 2. GFS Architecture

Why GFS has a strong scalability? Because metadata is small, because a main routine can control thousands of Chunk routines that store data.

4.3 What's more

In order to better support the extension, Google does not store any important state in the application server layer, but mainly persists the data in the datastore layer, so when the application traffic suddenly bursts, the expansion can be achieved by adding a new server to the application.

5. Transparency

5.1 Definition

Transparency is some aspects that a distributed system want to hide from the user. In other words, the distributed system should be a whole for users and applications, not a simple set of components that work together.

5.2 BigTable

The difficulty in achieving transparency is that creating a distributed database system provides an efficient means of accessing the sub databases at these nodes. And the Google App Engine uses a distributed structured data storage system called "BigTable".

BigTable is a mapping data structure supports multiple levels. It is also can be considered as a large-scale, fault-tolerant self-management system. This kind of system uses structured files to store data and have large memory and storage, which means it can process millions of "Read" and "Write" operations per second.

A multi-level mapping data structure is a sparse, multi-dimensional and sorted Map. Each Cell is three-dimensionally positioned by row keywords, column keywords, and timestamps. The content of the Cell is a string that is not interpreted. The BigTable data shows below:

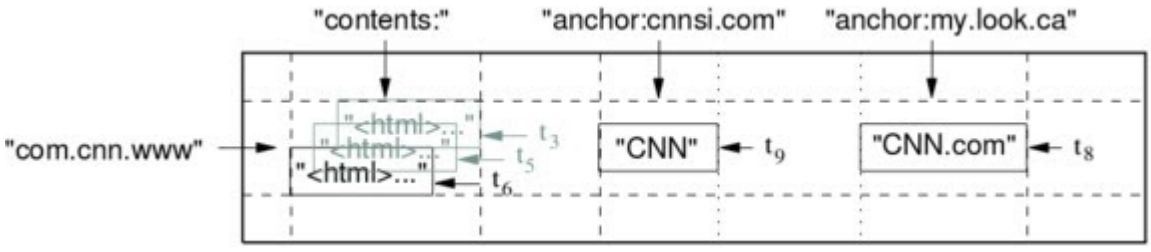

Figure 3. BigTable Data

In struct: BigTable is build on the GFS (Mentioned in **4.2**) and the Chubby (Mentioned in **6.3**). And inside BigTable, the structure could be divided into two parts: 1) Master node, process metadata-related operations and support load balancing. 2) Tablet node, store the fragmented tablet of

the database and provide corresponding data access. And the Tablet node is build on SSTable, which has good support for compression. And its struct diagram shows below:

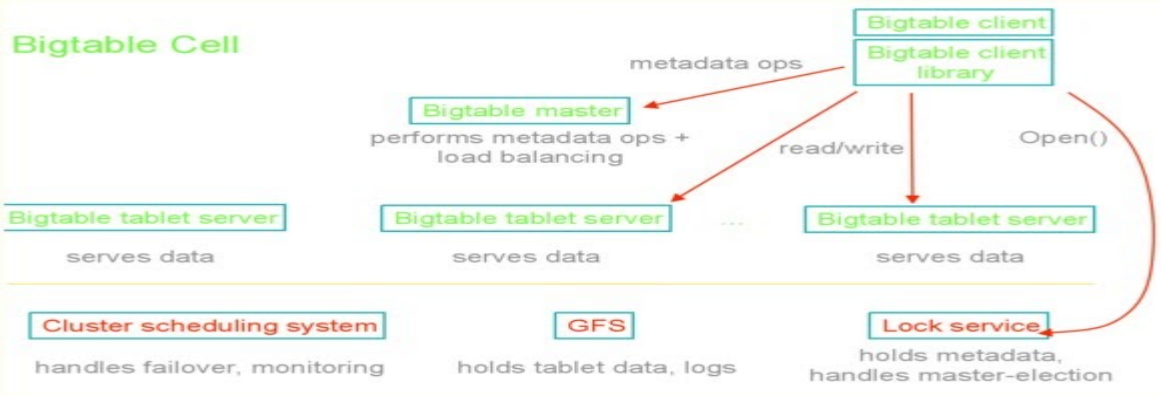

Figure 4. BigTable Struct

“BigTable” is serving a huge number of Google products and projects, including Google app engine. And we think “BigTable” is the key technology for Google app engine to achieve transparency.

5.3 User Interface

If we considered designing the transparency of distributed system as building a bridge. “BigTable” is the bridge pier and UI is the bridge deck. Relying on the GCP (Google Cloud Platform), Google App Engine has a powerful console on the webpage and also has a complete application on the mobile phone. When users use Google app engine, they only feels its integrity. It is like a whole complete application with multiple functionality, not some separate applications with different functionality. And we think it is important to have a good user interface, especially for a distributed system.

6. Openness

6.1 Definition

The openness of a computer system is the characteristic that determines whether the system can be extended and reimplemented in various ways. It could be considered as the degree (Easy to Difficult) to which new shared resources are added and used by various client programs.

6.2 A unified communication mechanism

As we mentioned in **3.2**, Protocol Buffers technology is how google app engine unified different kinds of supporting the addition and use of heterogeneous resources.

6.3 Access to shared resources

Google app engine use a distributed lock service called Chubby. Through Chubby, a huge number of clients in a distributed system can do "lock" or "unlock" operation for a particular resource, which is often used for the collaborative work of BigTable. The aspect is to achieve "locking" by creating files, and based on the Paxos algorithm of the famous scientist Leslie Lamport.

7. Conclusion

From the perspective of the design of distributed systems, the Google application engine is successful. It provides a good solution to the challenges of distributed systems. But that doesn't mean it's perfect, Google App Engine developers are often limited to the Google service framework and there are many limits under Google App Engine. If you enjoy Google's services, Google app engine is still a good choice.

8. Reference

8.1 https://en.wikipedia.org/wiki/Google_App_Engine

8.2 http://www.uml.org.cn/zjjs/201206193.asp

8.3 https://www.ejbtutorial.com/distributed-systems/challenges-for-a-distributed-system

8.4 https://cloud.google.com/appengine/?hl=zh-cn

8.5 CNKI: 罗黎霞.基于云计算的服务平台——Google App Engine[J].信息与电脑(理论版), 2009(08):93-94.

8.6 CNKI: 曹军委. 基于云计算的教学资源共享体系的研究与实现[D].安徽理工大学,2012.

8.7 CNKI: 林清滢,冯健文,陆锡聪.基于Google云计算平台的文件共享系统设计与实现[J].计算机时代,2014(07):23-25.