Scalability of architecture for large-scale software systems

Prajnya B Prabhu, Anusha M, Aparna Joshi, Anisha J Prasad, Nagaraj G Cholli

Department of Information Science and Engineering,
R V College of Engineering,
Bengaluru, India.

Abstract

Software systems today run on servers where multiple clients access the service. Large-scale software systems have different requirements like security, scalability, dependability, maintainability, reliability and performance. This paper focuses on scalability requirement of large-scale software systems. Scalability is the ability of the system to manage the growing load in a capable way. Comparison is done among different distributed file systems which are largely used. A difference between two caching mechanisms is brought out. Depending on the requirements and applications, appropriate caching technique and distributed file system can be implemented.

Keywords-Large-scale system; Scalability; file system; NoSQL; map-reduce; caching

I. INTRODUCTION

A large scale system is one which stores, accesses, processes a large amount of data, and contains numerous hardware elements and a large number of people involved. One of the major requirements of large-scale system is scalability. Scalability has two types; if a system is completely replaced by a new system which is better and powerful than the existing system, then it is known as vertical scalability and if the existing system is enhanced by adding more elements to it, then it is known as horizontal scalability. A good analogy is to view the large-scale system as a function which grows continuously over time. The complete architecture of a large-scale software system is not laid in the beginning because large scale software systems are evolutionary. Hence, as and when the requirements arise enhancements are made to the underlying components or new elements are added to the system.

Characteristics of large-scale system:

- Infinite duration of life
- changing flow of information and new sources keep joining
- Unknown, changeable, fluctuating boundaries
- highly complex

There are few basic components that are needed for a large-scale system. They are:

A. File systems

Storing and accessing files is based on the client/server architecture. In traditional systems, files were organized in a single hard disk drive or physical media. Distributed file systems (DFS) spread the files or the system's data over multiple disks or machines (nodes). The need to reduce fault tolerance of the node cluster, improve data consistency and the management of the metadata lead to different DFS designs. Most of the DFSs follow master- slave architecture, where one server is elected as the master server, which mainly consists of metadata, and a number of slave servers which consists of data. That is, the master server will have the information about the files and its mappings, and the slave servers will contain the contents of the file. DFSs uses data replication mechanism to fulfil availability and reduce fault tolerance. Appropriate locking mechanisms are employed for different DFSs to provide synchronization and valid data read and update.

B. Database

A large-scale system needs a database to store all its data. As large-scale systems are evolutionary, the system and the database need to be scalable. The relational database hinders data evolution which causes a problem while scaling across the clusters. In the past decade a database named NoSQL (Not only SQL) emerged. NoSQL databases are non-relational, open source, distributed and horizontally scalable databases [2]. Relational databases follow ACID properties: Atomicity, Consistency, Isolation and Durability whereas NoSQL follows BASE properties: Basically Available, Soft state and Eventual consistency. Basically Available means that NoSQL uses replication and sharding to prevent data unavailability. Soft state means that the state of the system may change after some time. Eventually consistent means that the data soon after the transaction may not be consistent, but later at some point of time, it will be consistent.

C. *Map-reduce algorithm*

Major application of map-reduce algorithm lies in the implementation of Hadoop frame work. This frame work is used for scaling the large systems. Hadoop has been developed to work with large clusters having more than thousands of nodes, and consists of two parts namely: Map Reduce and Hadoop distributed file systems (HDFS). HDFS is a highly scalable and reliable distributed file system, which is needed for managing large amount of data, and such file system is needed by map reduce algorithm to make to software system more scalable. The architecture of Hadoop consists of one master node and several slave nodes. The master node manages these slave nodes to store data, in addition to which it also plays a role in storing data. Master node usually saves Meta data which gives description about the organization of file system. Hence the capability of a cluster in HDFS is based on the memory size of the master node (Name node). The HDFS splits data into individual fragments and stores them in clusters. These clusters store each data fragment into two different racks. Two copies in one rack and one copy in a different rack. This way even if data from one rack is lost, it could be recovered from other racks, this addresses important feature, security of information.

D. Caching

Caching is an important parameter that enhances performance and scalability of large scale systems. Caching provides fast retrieval of data. Recently and frequently used data are stored in cache. In the hardware, cache memory is provided near the CPU and in the software, caching is used in browsers. Open-source distance memory caching technologies such as Memcached and Redis are used by many of the large scale systems worldwide. Distributed caching helps in scaling an application horizontally by reducing database access and making required data available locally in the cache. A commonly followed pattern is query result caching.

II. LITERATURE REVIEW

In Architecture of Large Scale Systems [1], the definitions and difference between traditional systems and large scale systems are given. The requirements for a large-scale system, taking Facebook as an example are discussed by the authors. The basic components which make up the large-scale system are file systems, databases, mapreduce algorithm, and caching. The file systems which are mainly used are distributed file systems. A new database called NoSQL is extensively being used for large-scale systems. Hadoop is a framework which uses mapreduce algorithm. Caching is a way to temporarily store data for fast retrieval. The paper discusses the horizontal scalability of large scale systems using the basic components.

NoSQL databases [2] are non-relational database management systems, but are basically derived from RDB database systems. NoSQL databases are used where scalability is one of the main aspects as they are designed to run on distributed systems. They follow BASE properties. The components and data store types required for data architecture in NoSQL are discussed briefly.

There are various available DFS designs out there and the paper [3] deals with the most important ones addressing the fault tolerance, data consistency and metadata maintenance of the file system. As the data and the computing ability increases, a robust file system with impeccable performance, availability, reliability and scalability are to be chosen. A brief analysis of the considered file systems are done and according to our requirements, the best among the lot is to be selected is what the author wants to convey.

HDFS results in a lot of energy consumption to keep the system running. The paper [4] introduces a new file system named GreenHDFS which consists of two data node zones, the cold zone and the hot zone. The cold zone consists of less frequently used files and the corresponding server would be in sleep mode by default. The hot zone makes up the more frequently accessed files. Due to the sleep mode of the cold zone data nodes, there is high energy savings of the system resulting in less energy costs.

The paper [5] describes the design and implementation of the Sector storage and the Sphere compute cloud. Sector storage is not only available to the data centres but it's also scalable to a geographically distributed data centres. Security mechanisms for DFS has become a major problem and this is addressed to a considerable extent by incorporating security servers that provide access to authorized clients only. The master node would check for the authenticity of the client and establish a connection with it for data transfer.

The map reduce for big data analysis [6], a distributed file system (DFS) to achieve scalability of data and utilization of map reduce algorithm to incorporate DFS is addressed. MapReduce implemented by Google is an open source frame work equivalent to Hadoop frame work. The data in DFS is partitioned and this data is represented as (key, value) pairs. In Map function, different partitions of data are processed and sorted in order, and the output of each (key, value) pair is merged using a unique hash function. The Reduce function is then invoked which for every key and its output is stored in DFS. The pre-processing is done using Map and post processing using Reduce.

Efficient analysis of big data [7], the difficulties in addressing big data and obtaining useful information from it is studied. This data is so large, the traditional databases and techniques cannot be applied. Analysing big data is a huge challenge, as it needs the researchers to recognize hidden patterns, relations and extricate useful information and stores them efficiently. HDFS is designed specially to handle big data and use it in drawing important conclusions, which in turn uses Map-Reduce algorithm for processing data.

Paper [8], deals with various caching issues from the pattern perspective. Primed caching pattern and demand cache pattern are the two main classifications. Primed is useful when a part of the data set can be predicted and demand, when it cannot be. The pros of both are combined in a flexible implementation.

The Google File System (GFS) [9] has been implemented at Google as the storage platform and for the processing of data. GFS is scalable, fault tolerant and runs on inexpensive commodity. The authors have given the detailed description of the architecture and have discussed the challenges faced during design, such as component failures, huge files, file mutation and co-designing applications.

III. COMPARISON BETWEEN DIFFERENT DISTRIBUTED FILE SYSTEMS

Scalability in file systems can be achieved by using distributed file systems. Some of the features of distributed file systems are reliability, availability, fault tolerance and efficiency. Four distributed file systems which are of interest are compared in TABLE I.

GFS consists of a master node that maintains metadata of all chunks and the mapping from the file to chunks. Whenever a client makes a request to the master node, the master node sends metadata to the client. The client would then establish a connection with the appropriate server for data transfer. The data is triple duplicated, so that when a chunkserver fails, the master node can redirect the client to other chunkserver which contains the duplicated data. The master contains all metadata such as access control information, mapping from files to chunks and the locations of the chunk servers. If the master fails, then all the chunkservers are scanned to obtain the metadata and a new master server is elected. HDFS is an open source version of GFS. There are different variants of HDFS such as Ring File System (RFS), GreenHDFS, Efficient Distributed File System (EDFS) and Quantcast File System (QFS) [3]. The General Parallel File System (GPFS) is a centralized DFS which is a parallel, shared disk file for cluster computers. It supports large amount of disks and distributed locking is used for synchronization. GPFS elects a metanode at the centre to manage data about file data. If a node fails, GPFS tries to restore the metadata from the failed node and frees all the resources captured by that node. The sector is a DFS that provides security mechanisms which the other DFS fails to. It consists of a master server, security server and a number of slave servers. The master and slave servers are the same like in GFS. The security server connected to the master is used to authorize the access to the distributed data. The master node can open a connection with a client who has an authorized access for the data transfer. Security servers are basically used to maintain user accounts, file access information and others.

IV. CACHING PATTERNS

Caching is a process where the recently accessed data are stored in and retrieved from the cache memory. Caching pattern describes how to store and reuse resources by effectively storing and retrieving in a faster way to improve performance. Two caching patterns- primed cache and demand cache are compared in TABLE II.

TABLE I COMPARISON BETWEEN DIFFERENT DISTRIBUTED FILE SYSTEMS

Name	GFS	HDFS	GPFS	Sector
Scalability	Scalable	Scalable	Extreme Scalability	Scalable
Performance	Good Performance	High Throughput	High Performance	High Performance (2-4 times faster than HDFS)
Fault Tolerance	Fault Tolerant	Highly Fault Tolerant	Low Fault Tolerant	File-system level Fault Tolerance
Architecture	Master slave	Master slave	Shared Disk File System	Master (one or more), Security server and number of slaves
Data Duplication	Data is triple replicated	Data is preferably replicated thrice in racks	Data is only duplicated by RAID	Data is triple replicated
Data set divided into	Fixed 64 MB chunk size blocks	Preferably 128 MB blocks	256 KB block size by default	Data is stored as File
Developer	Google	Yahoo	IBM	Dr. Yunhong Gu

TABLE II DIFFERENCES BETWEEN CACHING PATTERN

Pattern	Primed Cache	Demand Cache
Applicability	• Cache is populated with predicted set of data • Useful for predictable, frequently read data	• Cache is populated on demand i.e as and when data is requested by applications • Useful for unpredictable, frequently read data
Advantages	• Reduced data access overhead • Optimised cached data set	• Quick initialisation • Minimum cached data sets
Disadvantages	• Semantics involved in pre-populating • Possibility of memory leaks	• Slow cache population • Possibility of memory leaks

The Primed caching pattern populates the cache with a predicted set of data whereas the Demand caching pattern populates the data dynamically that is on demand. Pre-populating the data in primed cache will lead to fast retrieval of data but populating the data in the cache is expensive as the data needs to be carefully selected to be stored in the cache. In Demand cache, the data population is a slower process as they are saved in cache as and when data is requested.

V. CONCLUSION

Large scale systems can be referred as system of systems. Technology today is growing at a rapid rate and for a new emergent technology, the large scale system cannot be built from the scratch. The reason, being the time constraint. Because by the time we develop the system, some other new technology and requirements would arise. Hence the systems are built such that they are scalable in every dimension. Scalability can be best achieved by using distributed systems.

This paper gives an insight on the different components that can be used for the architecture of a large scale system. Some distributed file systems –GFS, HDFS, GPFS and Sector- are compared based on their design and features.

REFERENCES

[1] Arne Koschel, Irina Astrova, Elena Deutschkämer, Jacob Ester, Johannes Feldmann, “Architecture of Large-Scale Systems”, World Academy of Science, Engineering and Technology International Journal of Computer, Information, Systems and Control Engineering, 2013

[2] Vatika Sharma, Meenu Dave, “SQL and NoSQL Databases”, International Journal of Advanced Research in Computer Science and Software Engineering, ISSN: 2277128X, August 2012.

[3] Yuduo Zhou, “Large scale distributed file system survey”, Indiana University Bloomington.

[4] R.T Kaushik and M. Bhandarkar. “GreenHDFS: Towards an energy-conserving, storage efficient, hybrid Hadoop compute cluster”, Hot Power, 2010.

[5] Yunhong Gu, Robert L Grossman, “Sector and Sphere: The Design and Implementation of a High Performance Data Cloud”, University of Illinois at Chicago and Open Data Group.

[6] Kyuseok Shim, “Map-Reduce Algorithms for Big Data Analysis”, Seoul National University.

[7] Siddaraju, Sowmya C, Rashmi K, Rahul M, “Efficient Analysis of Big Data Using Map Reduce Framework”, DrInternational Journal of Recent Development in Engineering and Technology, June 2014.

[8] Octavian Paul ROTARU, “Caching Patterns and Implementation”, Computer Science and Engineering Department, "Politehnica" University of Bucharest, January-June 2006.

[9] Sanjay Ghemawat, Howard Gobioff, Shan- Tak Leung, “The Google File System”, Google, October 19-22, 2003.