A Brief Report on Distributed Storage System

Shi FENG 2007013247

May 5, 2011

Contents

1 Introducation

Distributed system is a major topic of current computer science. Among it, a distributed storage system usually takes advantage of multiple machines to gain capacity, reliability, availability, while for people who operates data, it seems that he is facing with a single machine and needn't care about the whole backend framework.

2 Categories

There are a lot of typical distributed storage systems. Google File System[9], MooseFS[3] and many others are classified as *distributed file system.* A client can communicate with the cluster to store and retrieve data much like operating files in a local file system. Usually, they support full namespace hierarchy and the data is accessed with a path.

Another category is classified as *distributed database.* The significant feature of the data stored in a distributed database is that the data is stored and accessed aligned by *columns* and *rows.* Sometimes, more strengthful databases also record relationship between data and support complicated query semantics.

Apart from the two categories mentioned above, a novel kind of distributed storage system is getting more and more popular. It is not only light-weighted, as it doesn't support relationship between data or just supports weak relationship, but also flat, usually because it doesn't support complicated namespace hierarchy but just a map of key-value pairs. On the opposite, the system which belongs to this category is usually of high performance, to be measured by throughput, latency, availability, reliability and other criteria. Such systems are called *distributed key-value stores*. Yahoo's PNUTS[7], Douban's BeansDB[1], Kyoto Cabinet[2] from Japan and Bitcask[10] are all successful distributed key-value stores. As such storage systems are light-weighted, some systems, like MemcacheDB[5], decide to put data in memory or virtual memory to gain performance burst.

3 Fundamental Concepts

Before I summarize existing distributed storage systems, I would like to explain some fundamental concepts related to distributed system. Without a clear introduction on these conceptions, it would be impossible to comprehend the beauty and elegance of architecture designs on famous distributed storage system.

3.1 Replication

Replication is copies of same data. In distributed system, data is distributed to many computing and storage machines. Under most cases, data is evenly stored on different machines. The world will be easy but fragile if we don't make copy of data. Let's take a closer look at why data backup is necessary with some simple calculation. If the possibility of failture for a single machine is p, and for simplicity, we assume all the machines are independently identical, i.e., the possibility of failure for each machine equals p and a machine never notices whether his buddy is alive or dead. What is the possibility that a system containing n identical machines goes down with one or more machine failing to work at some time? Yes, you are right! It is $1-(1-p)^n$. So what does this mean? Although p may be small, when n gets larger and larger, the result tends to reach 1! Commonly, a datacenter of companies like Google contains from thousands of to millions of machines with moderate disks. Disk failures are not rare but a common case. If the data is not replicated, it would be impossible to ensure the integrity of data, as recovery of data from a failed disk costs time, computation resource and network bandwith, yet this is not always possible. Hence making copies of data is critical in large systems like distributed key-value store.

3.2 Consistent Hashing

Replication of data may not be as simple as it seems at first glance. There are many tricky technologies behind to provide the correctness of the replication mechanism, and further the improvement of the performance. Situation becomes even worse when the system is distributed as we have to make decision on the choice of machine for different block of data.

Usually, the data is distributed by its key. First, a hash function is needed to calculate a hash value for the key. The key may be a string or even any binary stream, while the result of the hash function always falls into a finite range. The range, called *hash space*, is divided into several segments, each representing a machine. That machine takes responsibility of all the data with keys whose hash value is within that range.

Different distributed algorithms vary in their hash functions. A simple instance could be a distributed system which adopts *Cyclic Redundancy Check* as its hash function. As we mentioned above, each machine is associated with a segment in the range of hash value. However, in most cases, this range is further made up of several sub-segments, called hash slots. Often, hash slots within a segment are not located adjacently to each other in the hash space. Note that this is why the algorithm distributes data evenly among different machines, while the impact of a single machine failure is minimized.

Consistent hashing is a method to distribute data evenly within a storage cluster, regardless of the specific hashing function employed. If we concatenate the tail of the hash space to its head, the hash space will rewind like a ring, called the *hash ring.* We put some nodes on the ring and the ring is broken up into some arcs. These nodes are called *virtual nodes.* A physical node is a storage server, which is a collection of same amount of virtual nodes. The assignment of virtual node to physical one is random, which means that all the virtual nodes, which belong to the same physical node, may not be adjacent on hash ring. Different virtual nodes, which belong to different physical nodes, may not appear on the hash ring in a round-robin manner. They are just randomly distributed.

How to decide which storage server should take responsibility of the data associated with a specific key? If we don't take data replication into consideration, the server is the physical node found as follows. We traverse clockwise from the point on the hash ring representing the hash value of that key. The physical node, where the first virtual node met like this belongs to, is the storage machine which is responsible for that data.

When a machine refuses to work any more due to failure, the virtual nodes associated with it should be taken care by other physical nodes. It is obvious that each of such virtual nodes should share the same physical node, to which the next virtual node belongs, counting clockwise. As the virtual nodes conducted previously by the failed machine are distributed randomly, it is probably expected that the data handled previously by the failed machine will be distributed evenly across other active machines. It goes the same when a new machine joins the cluster.

In the real world, data is replicated into many copies. The data associated with a single key is stored on several machines, namely N physical nodes. These machines are determined by tranversing clockwise from the point on the hash ring representing the hash value of that key. The traversal stops when the first M virtual nodes belongs to N different physical nodes. These N physical nodes are the target machines.

4 Example Distributed Storage Systems

In this short article, I'll make a brief summary on several famous distributed storage system with introductions on their significant features.

4.1 Dynamo: Amazon's Highly Available Key-value Store

I believe the most famous distributed key-value store is Amazon's Dynamo[8]. Although it is neither open source nor available for organizations outside Amazon to use, it is highly recognized by scientists and engineers in the area of distributed computing.

Perhaps Dynamo gains popularity mainly because of its elegant design. It is a perfect example of minimizing system functionality to satisify basic requirements of application. Dynamo acts as an internal infrastructure for Amazon's many services, such as the on-line book stores. In most of the senarios, the service beyond Dynamo has such a requirement that data is highly writable. Considering this specific requirement, Dynamo is designed at first day to support high throughput and low latency of write request, while to sacrifice consistency hence increasing of read request both operation time and possibility of version conflict, which is tolerable within these services.

4.2 Bigtable: A Distributed Storage System for Structured Data

Bigtable[4] is an outstanding representative of the brand new *NoSQL*. It differs from traditional database by not supporting complex relationship between data, yet more flexible. Bigtable is considered as a multi-dimensional mapping of data. The last dimension is usually timestamp which means the database records historical snapshots of data. The database is flexible that dimension names of different data may be different.

4.3 Redis: An Open Source, Advanced Key-value Store

Redis[6] is an open source key-value store. It beats other hash tables for its rich type of values, such as binary stream, lists, sets, sorted sets and even hashes, while still maintaining high performance because all the data resides in memory. Redis not only performs data compression but also implements a virtual memory layer in user space to solve memory shortage. It is also fully journaled to enhance ability of fault tolerance.

References

[1] BeansDB. `http://beansdb.googlecode.com/files/Inside%20BeansDB.pdf`.

[2] Kyoto Cabinet. `http://fallabs.com/kyotocabinet/kyotoproducts.pdf`.

[3] MooseFS. `http://www.moosefs.org/`.

[4] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):1–26, 2008.

[5] S. Chu. MemcacheDB. `http://memcachedb.org/memcachedb-guide-1.0.pdf`.

[6] Citrusbyte. Redis. `http://redis.io/`.

[7] B. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. PNUTS: Yahoo!'s hosted data serving platform. *Proceedings of the VLDB Endowment*, 1(2):1277–1288, 2008.

[8] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review*, 41(6):205–220, 2007.

[9] S. Ghemawat, H. Gobioff, and S. Leung. The Google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.

[10] J. Sheehy and D. Smith. Bitcask. `http://downloads.basho.com/papers/bitcask-intro.pdf`.