Shard Level Transaction Based Cluster Management for Online Distributed Storage

Wei Liu[1.a] , Jiabao Zhao[2.b]

[1,2]Department of Management and Engineering, Nanjing University, Jiangsu, China [a]weiliu938@gmail.com, [b]jbzhao@nju.edu.cn

Keywords:cluster management, online distributed storage, shard level transaction

Abstract. Nowadays, distributed systems have been used everywhere, some for online applications, such as Apache HBase [1] and Cassandra [2], and some for offline applications, such as HDFS [3] and GFS. These systems achieve good result and largely reduce the cost. Due to the fact that distributed system is generally composed of lots of machines and failures occur recurrently, cluster management module is a must to handle these failures, thus provides availability and scalability. Low latency and high read and write performance is needed in many circumstances, such as in search engine and game field. Distributed storage for these systems need special ways to do cluster management. In this paper, a shard-level transaction based cluster management solution is proposed to handle such systems, and it can be easily extended.

Introduction

In IT field, especially in the field of the Internet, the increase of user generates a lot of data, which is of vital important for business. To store and make use of the data, a variety of distributed storage systems are designed and used. Such as Google's BigTable [5], MegaStore [6] and Spanner [7], Amazon's Dynamo [4] and Apache's Cassandra and HBase. These are all excellent distributed storage used in different circumstances.

According to business scenarios, distributed storage systems are broadly divided in two categories: offline systems and online systems. The goal of offline storage systems is high throughput, while online storage aims at low latency. Due to the fact that distributed storage systems are generally composed of large quantities of machines, failure is normal. To ensure availability, consistency and scalability, a cluster management module is needed to deal with all kinds of failures.

The biggest challenge of cluster management lies on reducing the impact to the cluster while doing failover, load balancing, data backup and other operations, especially for busy online system. In this paper, we propose a shard level transaction based cluster management solution for busy online storage systems in which low latency is vital. And the solution can expand easily to handle other kinds of failures.

Design and Solution

In order to solve various problems in cluster management, namely reducing downtime, minimizing recovery cost and the impact on system performance, we take several failure scenarios into our consideration.

Unstable network

For a busy online system, the network may be unstable due to the heavy bandwidth or CPU loads. If a network disconnection happens while doing failure detection, the related machine or shard will be considered a failure. However, there are cases the network recovers in the next second, thus unnecessary replications and switching operations that bring great costs to the cluster are triggered.

Replication control

Replication is a common means of achieving availability in distributed storage systems. However copying large quantities of data is likely to become the bottleneck of the whole system and make the system unstable.

Failure detection: Aggressive or Conservative

Generally specking, there are two strategies for failure detection: aggressive and conservative. Aggressive means that as soon as the failure is detected, the failover and repair operations start. However, conservative strategy always tries to make sure of the failure before doing any operation. We need an adjustable failure detection policy based on system load, usage scenarios and other aspects.

To solve the above problems, we propose the following cluster management module:

1. Failures are managed in terms of the shard.

2. There are multiple types of failures, each with different priority.

3. Failure on shard is a transaction; multi failures on the same shard are processed serially in priority order. Every transaction has a batch of states, persistent and atomic.

4. A dedicated state controller handles a kind of failure; shard controller handles all the state controllers.

The entire architecture is shown in figure 1.

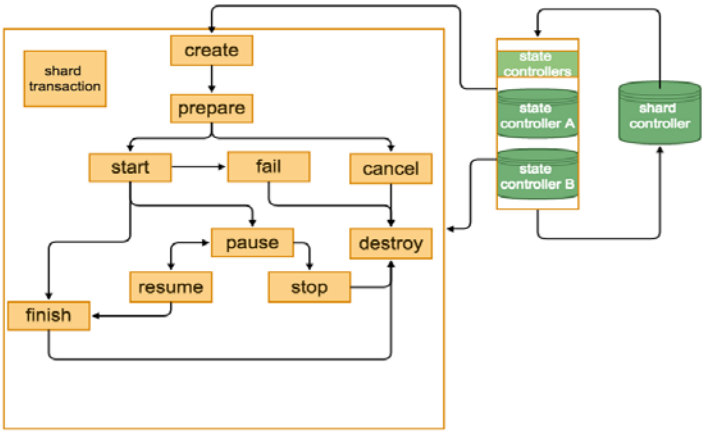

Fig. 1

In Figure 1, all the states in the left box are transaction states, and state controller can transform them with information collected from nodes and shard controller. Every kind of state controller deals with a kind of failure. Shard controller detects failure and monitors the entire cluster. The whole process belongs to a shard level transaction, in which every state is atomic.

Implementation

We use zookeeper [8] to persist state transaction in case of the failure of cluster management module. When the cluster management module crashed, another cluster management module takes over the management with the persistent state data on zookeeper.

We implement different kinds of state controller to handle different kinds of failures. Every state controller implements a group of state transition functions and several utility functions.

Shard controller mainly does three kinds of things: adjusting failure detection strategy, collecting cluster information for state controller and monitor cluster network state. We design these three main methods as plugins.

Result

We use our cluster management module in an online distributed storage system, which stores data for a search engine. The average delay requirements of the system is within 10ms and 95% long tail delay is no more than 50ms. The system consists of about 1,000 machines. The system is master-slave architecture, as shown in figure 2.

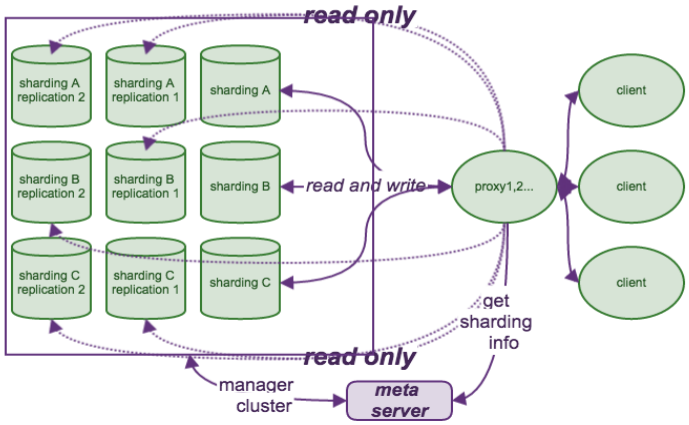

Fig. 2

The system is currently dealing mainly with two kinds of failure: failover and backup. Failover is a kind of failure, which caused by master downtime in a shard; Backup is another kind of failure, which caused by instance numbers not enough in a shard.

Failover state controller

As the system can at most tolerate 10 seconds of downtime per shard, we must carefully determine the timeout period. Smaller timeout period can reduce the total recovery time, but it will lead to higher error ratio of failure detection. Detailed results are shown in table 1.

Table 1 Failover timeout versus recovery time and error ratio

Timeout[s]	Average recovery time[s]	Average error ratio [%]
5	7.6	0.24
7	9.3	0.21
9	11.0	0.18

Backup Failure

Dealing with backup failures is more complicated, because it usually leads to a lot of data replication or data move. The backup state controller needs to calculate the cost of ongoing data operation during the whole lifecycle of the transaction to determine whether to pause or stop the fix process or not. For example, when a failure machine recovers, the backup state controller may pause the ongoing data replication if synchronizing the recovery machine costs less.

Our system shows that there is about 0.5% false positive failure test and 15% failures can recovery themselves in a busy system. Our cluster management module helps saving a lot of bandwidth and disk IO compared with the original module, which is based on redis sentinel.

Conclusion

Through our shard-level transaction based cluster management solution, the failure cost and recovery time of online distributed storage systems is cutting down. And we also achieved scalability. We test our solution in a real world cluster and obtain good results.

However, the solution mentioned here is mainly to deal with the failures within the same datacenter. Recently, several companies have proposed fast and efficient data replication across datacenters, such as Google Spanner[7]. For cross datacenter cluster, the difficulty lies on the limited bandwidth, which will cause long tail delay and slow replication. Further work will focus on the management of cross datacenter cluster.

Acknowledgements

* Thanks to the support of National Natural Science Foundation of China(70971063)
* bcorresponding author

References

[1] HBase, Apache. "A Distributed Database for Large Datasets." The Apache Software Foundation, Los Angeles, CA. URL http://hbase. apache. org.
[2] Cassandra, Apache. "The Apache Software Foundation." URL: http://cassandra. apache. org/(visited on 01/05/2013).
[3] Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." ACM SIGOPS Operating Systems Review. Vol. 37. No. 5. ACM, 2003.
[4] DeCandia, Giuseppe, et al. "Dynamo: amazon's highly available key-value store." ACM SIGOPS Operating Systems Review. Vol. 41. No. 6. ACM, 2007.
[5] Chang, Fay, et al. "Bigtable: A distributed storage system for structured data." ACM Transactions on Computer Systems (TOCS) 26.2 (2008): 4.
[6] Baker, Jason, et al. "Megastore: Providing Scalable, Highly Available Storage for Interactive Services." CIDR. Vol. 11. 2011.
[7] Corbett, James C., et al. "Spanner: Google’s globally distributed database." ACM Transactions on Computer Systems (TOCS) 31.3 (2013): 8.
[8] Hunt, Patrick, et al. "ZooKeeper: Wait-free Coordination for Internet-scale Systems." USENIX Annual Technical Conference. Vol. 8. 2010.