Cassandra to back applications

Mengchen Yu
Indiana University Bloomington
yumeng@indiana.edu

27 March 2011

Abstract

All data intensive applications require databases to back them. As modern applications have different requirement on performance aspects, the traditional RDBMS on single server is no longer a one fit all solution. Distributed database evolves as an alternation which fits the need of a fast pace development of applications. This paper picks Cassandra as a typical example of these new database systems, and by describing of Cassandra to give an inner view of distributed databases.

1 Introduction

Databases are crucial for applications that need to take record of huge amount of data and utilize the data for prediction, description or classifications. In the past, all of the data is stored in and managed by the database management system installed on a single host. When the amount of data accumulate day by day, the performance of this kind of database becomes disappointing. Also, when by chance the server quit work, it can be a disaster.

To solve these problems, distributed databases are introduced. Firstly, by maintaining a number of replications among the nodes, distributed databases strengthened the availability of data. Secondly, using distributed algorithms such as MapReduce, distributed databases achieve a high speed performance over traditional single-host databases.

Among the distributed databases, Cassandra is a typical one. It offers a simple data model that support dynamic control of data layout and format. It's designed to run on cheap commodity hardware and handle write throughput while not sacrificing read efficiency.

This paper will look into more details of Cassandra including the background of where Cassandra comes from and its initial mission, data model of Cassandra, Cassandra APIs. Above is in chapter 2, Cassandra overview. In chapter 3, there is a detailed discussion about Cassandra architecture. After that, we will review some test on performance and scalability of Cassandra as well as industrial examples using Cassandra to back applications.

2 Cassandra overview

This section gives an overview to Cassandra. The first part is the history of Cassandra. The second and third part describes Cassandra data model and APIs respectfully.

2.1 Background

Cassandra originated at Facebook in 2007 to solve that company's inbox search problem, in which they had to deal with large volumes of data in a way that was difficult to scale with traditional methods. The code was released as an open source Google Code project in July 2008. During it's tenure as a Google Code project in 2008, the code was updateable only by Facebook engineers, and little community was built around it as a result. So in March 2009 it was-

moved to an Apache Incubator project, and on February 17, 2010 it was voted into atop-level project[6].

2.2 Cassandra data model

2.2.1 Cluster

The outermost structure of Cassandra is the cluster, or ring, because Cassandra allocates data to different nodes in the hosts in a cluster by arranging them in a ring. Each node holds a different range of data. There are replications in the ring to ensure availability. If one node is down, other node with a replication of data can response to queries originally targeted on the down node.

2.2.2 Keyspace

A cluster is a container of keyspaces. In one cluster, there are usually one keyspace. Keyspcaces are the outermost container of data in Cassandra. There are three attributes for a keyspace: replication factor which means how many replications for a piece of data will exist over the whole cluster, replica placement strategy which indicates how the replica of data will be located in the ring and column families which is the sub level of data structure of Cassandra. There are at least one and always many column families in a keyspace.

2.2.3 Column families

There are two kinds of column families in Cassandra: simple and super. As the simple column family is used to contain columns which store the exact data, the super column family is used to contain simple column families. The benefit of using a super column family is to allow for nesting.

2.2.4 Columns

Columns are the basic structure in Cassandra to store the exact data. As the column families, there are two kinds of columns in Cassandra: simple columns and super columns. A super column is a special kind of column. Both kinds of columns are name/value pairs, but a regular column stores a byte array value, and the value of a super column is a map of subcolumns (which store byte array values). Each column family is stored on disk in its own separate file. So to optimize performance, it's important to keep columns that are likely to be queried together in the same column family, and a super column can be helpful for this.

2.3 Cassandra APIs

Up to now, Cassandra project provides a variety of high-level clients to support a quick deployment of Cassandra solutions. The supported languages are: Python, Java, Grails, .NET, Ruby and PHP. The clients are established on the basis of thrift APIs.

3 Cassandra architecture

In this section, we look into the architecture of Cassandra to see how it works. Basically we consider the peer-to-peer design and its corresponding gossip protocol. Also we talk about other issues in the implementation of Cassandra.

3.1 Peer-to-peer

In many other distributed data storage solutions, nodes are divided into masters and slaves.[2] There is single node failure threat to this approach. The Cassandra employs a peer-to-peer structure so that each node is identical to the others. As there are replications in the ring, if one node fails, the service will remain with a minor degrade of the performance. Data on the failed node can still be accessed.

The peer-to-peer design also makes it easy to scale Cassandra by adding new nodes. Because the behavior of each node is identical, in order to add a new server, you simply need to add it to the cluster.

3.2 Gossip and failure detection

To support decentralization and partition tolerance, Cassandra uses a gossip protocol for intra-ring communication so that each node can have state information about other nodes. The gossiper runs every

second on a timer. Hinted handoff is triggered by gossip, when a node notices that a node it has hints for has just come back online[3]. Anti-entropy, on the other hand, is a manual process; it is not triggered by gossip.

The gossip protocol in Cassandra is primarily implemented by the org.apache.cassandra.gms.Gossiper class, which is responsible for managing gossip for the local node. When a server node is started, it registers itself with the gossiper to receive endpoint state information.

3.3 Anti-Entropy and Read Repair

Anti-entropy is the replica synchronization mechanism in Cassandra for ensuring that data on different nodes is updated to the newest version. During a major compaction, the server initiates a TreeRequest/ TreeReponse conversation to exchange Merkle trees with neighboring nodes. The Merkle tree is a hash representing the data in that column family. If the trees from the different nodes don't match, they have to be repaired in order to determine the latest data values they should all be set to. This tree comparison validation is the responsibility of the org.apache.cassandra.service.AntiEntropyService class. AntiEntropyService implements the Singleton pattern and defines the static Differencer class as well, which is used to compare two trees; if it finds any differences, it launches a repair for the ranges that don't agree.

3.4 Staged Event-Driven Architecture (SEDA)

Cassandra implements a Staged Event-Driven Architecture (SEDA) which is a general architecture for highly concurrent Internet services.

In a typical application, a single unit of work is often performed within the confines of a single thread. A operation, will start and end within the same thread. But the case with Cassandra is different: its concurrency model is based on SEDA, so an operation may start with one thread, then the thread hands off the work to another thread, and so on.

But it's not the current thread to determine whether or not to hand off the work to another thread. Instead, work is divided into stages, and the thread pool associated with the stage determines execution. Rather than the whole operation, a stage is basic unit of operation, and a single operation is consists of stage after stage.

Because each stage can be handled by a different thread pool, Cassandra have a great performance improvement. This SEDA design also means that Cassandra is better able to manage its own resources internally because different operations might require disk IO, or they might be CPU-bound, or they might be network operations, and so on, so the pools can manage their work according to the availability of these resources.

4 Test to Cassandra

There are already many existing papers talking about performance test of Cassandra. In these papers, the performance is always measured from aspects as listed: write speed, read speed, failure tolerance, scalability and elastic speed up. The variables are always number of nodes (the extent of scale) and the data size (workload). The performance of Cassandra is always put into comparison with performances of other storage platforms such as MySQL, HBase, and PNUTS, etc.[5] Among the comparisons, the distributed solutions always have a better overall performance over traditional relational databases when the requirement of ACID and also features of relational model is low. Among the distributed solutions, Cassandra is very good at write/read intensive tasks.[7] As there are so many factors which can be tuned during the configuration of those platforms, such as replication factors and replication placement policies, it's hard to say which one is definitely better than one else. But Cassandra is really good at some kinds of tasks.

5 Industrial example

Cassandra is a project that initiated by Facebook whom want to have a distributed storage system for managing very large amounts of structured data spread out across many commodity servers, while providing highly available service with no single point of failure for their inbox search feature.

There are two kinds of search features, term search and interactions. Given the name of a person return all messages that the user might have ever sent or received from that person. In order to make the searches fast Cassandra provides certain hooks for intelligent caching of data. For instance when a user clicks into the search bar an asynchronous message is sent to the Cassandra cluster to prime the bu?er cache with that user's index. This way when the actual search query is executed the search results are likely to already be in memory.

6 Conclusion

In this paper we look into the history and some implementation detail of Cassandra. Then we have a review of the tests that have been done in comparison of Cassandra and other storage systems. Finally a example of Cassandra being used by Facebook is illustrated.

7 References

[1] Apache Cassandra Project official website, http://cassandra.apache.org/

[2] Avinash Lakshman Prashant Malik,Cassandra - A Decentralized Structured Storage System, ACM SIGOPS Operating Systems Review archive Volume 44 Issue 2, April 2010.

[3] Ken Birman, The promise, and limitations, of gossip protocols, ACM SIGOPS Operating Systems Review Volume 41 Issue 5, October 2007.

[4] Matt Welsh et al., SEDA: An Architecture for Well-Conditioned, Scalable Internet Services, In Proceedings of the Eighteenth Symposium on Operating Systems Principles (SOSP-18), Banff, Canada, October, 2001.

[5] Brian F. Cooper et al., Benchmarking Cloud Serving Systems with YCSB, Proceedings of the 1st ACM symposium on Cloud computing,2010.

[6] Wikipedia for Apache Cassandra, http://en.wikipedia.org/wiki/ApacheCassandra

[7] Yingjie Shi et al., Benchmarking Cloud-based Data Management Systems, CloudDB10, October 30, 2010, Toronto, Ontario, Canada.

[8] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazons Highly Available Keyvalue Store. In Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, ACM Press New York, NY, USA, 2007.