Decentralized storage system-Cassandra

Jiang Wu
Indiana University Bloomington
wu44@indiana.edu

26 March 2011

Abstract

Cassandra is a distributed storage to store a large amount of structured data for the distributed system. It aims to run on the top of the infrastructure of thousands nodes and spread the data cross many servers [1]. At this scale level almost all kinds of components fail continuously. The way Cassandra manages the persistent state in the face of these failures drives the reliability and scalability of the software systems relying on this service. Although Cassandra resembles a database and support on many data design and implementation strategy, it is not fully relational. It is semi-structured database use key-value storage. Cassandra provides client this simple data model to support dynamic control over a large scale of data. What is more, it does not require an expensive hardware and do a high efficiency on writing without sacrificing any read operation, the configuration of using Cassandra is also much easier compare to other distributed semi-structure database.

1 Introduction

To meet the scalability and reliability needs Facebook has developed Cassandra. The scalability and reliability is described as below:

Facebook runs the most significant and largest social networking platform which serves hundreds of millions users at the same times through thousands of servers located in different data centers around the whole world. So there are strict requirements for operations on Facebook's platform such as reliability, efficiency and performance, and because of the continuous growth of data and users the platform of Facebook needs to be deployed in to highly scalable. There are always some server and network components that are failing at any given time when dealing with failures in an infrastructure comprised of thousands of components is our standard mode of operation. So it requires the software systems to be constructed that can treat failures as the normal thing rather than the exception [2].

Cassandra is designed to deal with the large scale of data on a distributed system and have a high efficiency on writing because take Facebook as an example, Facebook platform is required to handle a very high write throughput like billions of writes per day, and also the number of users is a large scale. And actually the users are served from data centers which are distributed in different area on earth geographically, so the key is being able to replicate data across data to keep search latencies down. Cassandra is now deployed as the backend storage system for multiple services within Facebook and doing well for such goal so far[3].

2 Architecture and method for Cassandra

2.1 Data model

A table in Cassandra is a distributed hash table which is multi-dimensional map indexed by a key value[4]. There is no limitation for the number of rows because

it supports for large scale of data. The row key in a table is a string with no size restrictions, although typically 16 to 36 bytes long. Every operation under a single row key is atomic for each replica no matter how many columns are being read or written into. Columns are grouped together into sets called column families which are similar to what happens in the Bigtable system. We introduce the most important concepts involved.

Column Family: A Column Family is the unit of abstraction containing keyed rows which group together columns and super columns of highly structured data. Column families have no defined schema of column names and types supported. Furthermore, columns are added only to specified rows, so different rows can have different numbers of columns in any given family [1]. This is in stark contrast to the typical relational database which requires predefined column names and types.

Column: A column is the atomic unit of information supported by Cassandra and is expressed in the form name:value.

Super Column: Super columns group together like columns with a common name and are useful for modeling complex data types such as addresses other simple data structures.

Row: A row is the uniquely identifiable data in the system which groups together columns and super

Keyspace: The Keyspace is the top level unit of information in Cassandra. Column families are subordinate to exactly one keyspace.

2.2 Query method

Because the non-relational model for Cassandra it cannot support the complicated quires such as join, aggregation, and so on. It can only provide simple queries: Insert, Get, and Delete. Although Cassandra internally does not support query language, it support range queries which allow users to batch primitive operations and simplify query programing. (i.e. we can use range query to implement Select and Group by) . The Cassandra data store is implemented in Java. However there is no Java API for communicating with Cassandra from a separate address space only command line can be used. Cassandra implements services using Thrift which is a framework that includes a high level grammar for defining remote objects, types and servers[5]. What is more, variety of language can be used to program the code generator that produces client and server MRI. Each Cassandra node starts a Thrift server exposing services for interacting with data and introspecting information about the cluster. The Thrift API to Cassandra leaves open the possibility of a perceived single point of failure and does not intelligently route service invocations to replica nodes.

2.3 Partition

One of the key design features for Cassandra is the ability to easy-scale-up so this requires the ability to dynamically partition the data over the set of nodes in the cluster. Cassandra is more like a distributed hash table which partitions data across the cluster not only use consistent hashing but also uses an order preserving hash function to do so. In consistent hashing the output range of a hash function is treated as indexed circular space like a "ring" structure. Distributed (DHT method) hash table provides a scalable alternative to the central server for distribution to see more than counterparts without the necessary coordination number of central and storage [6]. DHT nodes stores information about the key contains 0 or more other nodes in the system range of information available and the key range. The more information each node maintains about its neighbors, the fewer hops required to get to the correct node. Typically there exist two ways to address this issue: One is for nodes to get assigned to multiple positions in the circle (like in Dynamo), and the second is to analyze load information on the ring and have lightly loaded nodes move on the ring to alleviate heavily loaded nodes as described in. In Cassandra each server is assigned a unique token which represents the range keys for which it will be responsible for [7]. First, the random position assignment of each node on the ring leads to non-uniform data and load distribution. Then, the basic algorithm is oblivious for the performance of nodes [8]. Cassandra opts for the latter as it makes the design and implementation very tractable and helps to make very deterministic choices about

load balancing.

2.4 Replication

To allow keys to be read and written even when a responsible node has failed, Cassandra will keep N copies distributed within the cluster for fault tolerance [9]. Higher values of N contribute to availability and partition tolerance; however it need more for read consistency of the replicas. For each token in the ring, the replication strategy returns N?1 optional endpoints [10]. The coordinator is in charge of the replication of the data items that fall within its range. In addition to locally storing each key within its range, the coordinator replicates these keys at the N-1 nodes in the ring. Cassandra provides the client with various options for how data needs to be replicated. Cassandra provides various replication policies such as
Rack Unaware",
Rack Aware" (within a datacenter) and
Datacenter Aware". Rack unaware is the default strategy used by Cassandra which ignores the physical cluster topology. This strategy begins at the main node for a token and returns the endpoints of the next N ?1 nodes in the token ring.. This design can make the system to continue operating even if a rack or whole data center being crashed or unavailable.

2.5 failure detection

In Cassandra failure detection is used to avoid attempts to communicate with unreachable nodes during various operations. Cassandra uses a modified version of the Accrual Failure Detector of which the failure detection module set as a value which represents a suspicion level for each of monitored nodes[11]. The basic idea is to express the value of on the distributed system which is dynamically adjusted to corresponding to network and load conditions at the monitored nodes. Every node in the system maintains a sliding window of inter-arrival times of gossip messages from other nodes. The distribution of these inter-arrival times is determined and is calculated.

3 Compare Cassandra to other databases

Compare Cassandra to MySQL, For basic queries (like Delete, Get), increasing the number of nodes within a small amount of number, MySQL performs significantly better than Cassandra. It is because Cassandra is designed for large amount distributed data. So when the data is not large enough or not well distributed we cannot tell the advantages compared to relational database like MySQL.

Compare Cassandra to Hbase, they are all distributed key-value store for large amount data. When the node is within a small numbers they all perform not well however Hbase is a little bit better[12].. But when the distributed system is getting deployed and nodes and data getting larger Cassandra performance is better than Hbase especially in the I/O time. Because Cassandra supports a high efficiency of writing without sacrificing the reading time and it can treat some failure like norm instead of exception because of its failure detection[13].. Besides that the configuration for using Cassandra is much easier than configure for Hbase.

4 Implementation

The Cassandra process on a single machine is primarily consists of the following abstractions: partitioning module, the cluster membership and failure detection module and the storage engine module. Each of these modules has been implemented from the ground up using Java. The cluster membership and failure detection module is built on top of a network layer which uses non-blocking I/O. All system control messages rely on UDP based messaging while the application related messages for replication and request routing relies on TCP. The request routing modules are implemented using a certain state machine. When a read/write request arrives at any node in the cluster the state machine morphs through the following states (i) identify the node(s) that own the data for the key (ii) route the requests to the nodes and wait on the responses to arrive (iii) if the replies do not arrive within a configured timeout value fail

the request and return to the client (iv) figure out the latest response based on timestamp (v) schedule a repair of the data at any replica if they do not have the latest piece of data.

The Cassandra system indexes all data based on primary key. The data on disk is broken down into a sequence of blocks. Each block contains at most 128 keys and is demarcated by a block index. The block index captures the relative o_set of a key within the block and the size of its data. When an in-memory data structure is dumped to disk a block index is generated and their o_sets written out to disk as indices. This index is also maintained in memory for fast access. A typical read operation always looks up data _rst in the in-memory data structure. If found the data is returned to the application since the in-memory data structure contains the latest data for any key. If not found then we perform disk I/O against all the data _les on disk in reverse time order.

5 Conclusion

Cassandra is built, implemented, and operated as a storage system which providing scalability, high performance, and wide applicability for distributed system. It can support a very high update throughput while delivering low latency. It giving the high efficiency by acting like distributed hash table to do the partition and replication. It treats the some failure as norm instead of throwing exception for a better failure tolerance. It is getting well used among a lot of distributed system.

6 References

[1] Cassandra. http://en.wikipedia.org/wiki/Apache_Cassandra. Retrieved on Oct 12, 2010.

[2] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A Distributed Storage System for Structured Data. OSDI'06: Seventh Symposium on Operating System Design and Implementation, 2006, Seattle, WA, 2006.

[3] 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.

[4] A. Lakshman and P. Malik. Cassandra: A Decentralized Structured Storage System. In Proceeding of 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware, 2009.

[5] M. Slee, A. Agarwal and M. Kwiatkowski. Thrift: Scalable Cross-Language Services Implementation Facebook, Palo Alto, CA, 2007.

[6] Cassandra Wiki http://wiki.apache.org/cassandra/FrontPage.

[7] A. S. Tanenbaum, M. V. Steen. Distributed Systems: Principles and Paradigms. 3rd edition. Prentice Hall. 2008.

[8] C. Bunch, J. Kupferman, and C. Krintz. Active Cloud DB: A Database-Agnostic HTTP API to Key-Value Datastores. In UCSB CS Technical Report 2010-07.

[9] Cassandra. http://cassandra.apache.org/.

[10] R.Cattell, Datastore Survey,in progress, Apr. 2010. http://cattell.net/datastores/Datastores.pdf.

[11] X. Dfago, P. Urbn, N. Hayashibara, T. Katayama. The accrual failure detector. In RR IS-RR-2004, Japan Advanced Institute of Science and Technology, pages 66-78, 2004.

[12] Cassandra UseCase. Available at http://wiki.apache.org/cassandra/UseCases.

[13] Cassandra FAQ. Available at http://wiki.apache.org/cassandra/FAQ.