Conflict Resolution in Cassandra

Milind Parikh
milindparikh@gmail.com

Overview

Any distributed database that makes an tradeoff between availability and consistency in context of CAP and chooses availability over consistency will face potential conflicts in data. This is a given. However the approaches taken to resolve conflicts defer from product to product. It is the purpose of this article to provide a framework under which the different approaches can be analyzed. This article uses Cassandra 0.7.

Introduction

The reader is expected to be familiar with the CAP theorem and the context of operation in CAP. Briefly, in a distributed data scenario, the CAP theorem states that one can choose only two out of three choices : Consistency, Availability and Partition Tolerance. In a distributed scenario, network partitions happen all the time; given enough failure points. Therefore Partition Tolerance is almost a requirement for modern distributed databases. Therefore one is really limited to choose between Consistency and Availability. This article is interested in the class of products where Availability is valued over Consistency.

In this class of products, the term "eventual consistency" is almost always used. The term, broadly speaking, means that the system will eventually converge to consistency; but consistency is not immediate. I will defer the actual analysis of eventual consistency; but in some cases eventual consistency can be achieved in 100s of milliseconds. So the term "eventual consistent" may not be as bad as it sounds.

Assuming, without the loss of generality, that the system consists of three nodes on which writes/reads are happening. I label these two nodes as X, Y & Z. These three nodes can be geographically seperated. In the normal mode of operation, these three nodes can communicate with each other at normal network speeds. Alice and Ben are two business users who work on the system. Their assumption of the system is that it should be 'always' available; even when the communication channel between the nodes breaks down. We introduce the data element 'D' on which both Alice and Ben are working.

At the begining , 'D' has an attribute called 'count'. Alice assigned the count to value 1. She assumes that Ben will see the value of count to be 1. In a connected system, all nodes will see the same value of 1 within the network latency limits. We assume that both Alice and Ben are not superhumans with reflexes that are greater than the latencies within the system. Of course, how these three nodes will 'see' the same value will depend on the implementation.

Now assume that both the nodes become disconnected. Alice writes to 'count' the value of 2. She assumes that Ben got the value of 2. Ben, on the other hand, writes 4 to the value of count. Ben assumes that Alice got the value 4. This happens because both

Alice and Ben are writing to different nodes; X & Y respectively. Then X & Y are connected backup. At this point, there is a conflict. What is the value of D's count variable ? Is it 4 because Ben is Alice's manager? Is it 2 because Alice has more details about the business processes that revolve around setting the value of count? The point is that there is a confict and it must be resolved. The system has no way of knowing what the "right" choice is. It may make a choice; but not necessarily the "right" one.

Analysis of Cassandra

Background

Keys

A key is the basic data element against which a set of values can be stored. A key can be stored in one or more places. Such a place is called a replica.

A keyspace is the mechanism through which a set of keys can express certain common system properties. A keyspace can designate whether it wants to be written to one replica or multiple replicas. This is called the Replication Factor. A replication factor of 2 means that the key will be written to two replicas.

Node

A node is a server element where a set of keys can be stored. Cassandra system usually consists of multiple nodes.

Writes

Write is the mechanism through which data gets into the system. It is completely conceivable that the write may be only partial successful. There is no concept of rollback in Cassandra. If the write is partially successful, the system will try, in the backrgound, to make the write successful. The meaning of success is defined by the consistency levels as defined below.

Reads

Read is the mechanism through which data is retrieved from the system.

Read Repairs

Read Repair is the mechanism through which the most recent version of the key is pushed out to any out-of-date replicas;when a read is made against the specific key. RR can be made in the background or before returning the data.

Hinted Handoffs

Hinted handoffs is the ability to write the key to any node; including a node that is NOT a final destination for any replicas.

Replication Factor

Replication Factor (RF) is the number of nodes (N) to which the values of a key are written to; eventually.

Consistency Level

Consistency Level (CL) is the specification of how reads and writes behave in context of nodes in a cluster. CL is treated differently in a read versus a write.

LEVEL	**READ**	**WRITE**
ZERO	NA	Completely Asycnchronous
ANY	NA	Write to at least one node; including hinted handoff recipients
ONE	Return the record fetched by the first replica and initiate RR	Write to at least one replica's commit log and memory table before returning
QUORUM	Return the record with the most recent timestamp in the majority of the replica's (N/2 + 1) reporting	Write to N/1 + 2 replicas before responding to client
LOCAL_QUORUM	Return the record with the most recent timestamp in the majority of the replica's	Write to RF/2 + 1 replicas within the local data center

	in the local data center reporting	
EACH_QUORUM	Return the record with the most recent timestamp in the majority of the replica's in each data center reporting	Write to RF/2 + 1 replicas within the each data center
ALL	Return the record with the most recent timestamp once all of the replicas have replied. Any non-responsive replicas will fail the read.	Write to all N replicas before responding to client. Any non-responsive replica will fail the write.

* Cassandra guaranties that if you do a (succeeding) write on W replica and then a (succeeding) read on R replica and if R+W>N, then it is guaranteed that the read will see the preceding write. This is eventual consistency.

* A { WRITE (CL=QUORUM) = success } followed with { READ (CL=QUORUM) = success } will always see the preceding write.

Detailed Analysis

Assume a RF = 3. In case of Cassandra, time sequencing is pretty important.

time	actor	action	description
t0		All nodes are connected	
t1	Alice	WRITE (CL=QUORUM) : D{'count' = 1}	w1
t2	system	w1 succeeds	in milliseconds is written to two nodes -- say X & Y
t3	system	All nodes (because RF = 3) have the data	in milliseconds

t4	Ben	READ (CL=QUORUM) : D{'count' = 1}	read from two nodes X & Y Y & Z X & Z
t5	Alice	READ (CL=QUORUM) : D{'count' = 1}	read from two nodes X & Y Y & Z X & Z
t6	**system**	**X disconnects from Y & Z**	**Split brain :** **Alice primary write is through X** **Ben primary write is through Y & Z**
t7	Ben	WRITE (CL=QUORUM) : D{'count' = 4}	**w2 succeeds.** Q = 2 w2 cannot be propagated to X... spawns process in background
t8	Alice	WRITE (CL=QUORUM) : D{'count' = 2}	w3 fails... **but data is written into X.. conflict is created; but system is not aware about the conflict**
t9	Ben	READ (CL=QUORUM) : D{'count' = 4}	read from 2 nodes Y & Z
t10	Alice	READ (CL=QUORUM) : Read rejected	cannot read from two nodes
t11	**system**	**X, Y & Z are connected backup**	
t12	Alice	READ (CL=QUORUM) : D{'count' = 4}	Read from 2 nodes X & Y Y & Z X & Z choice was Y & Z RR initiated in background
t13	system	RR sees the new value posted by Alice on X.	automated conflict

		Y-> D{'count' = 2} Z-> D{'count' = 2}	resolution by timestamp
t14	Alice	READ (CL=QUORUM) : D{'count' = 2}	Read from 2 nodes X & Y Y & Z X & Z choice was Y & Z

Summary

1. Cassandra converges to the newest value of the key on an automated basis.
2. Under Quorum, Cassandra guarantees that once a read has seen a write, all others will see that same write.
3. It is possible in Cassandra to have updates being silently dropped. i.e Ben's update of 4 was dropped without having a choice.