Why Cassandra doesn't need vector clocks

BY JONATHAN ELLIS - SEPTEMBER 2, 2013 | 11 COMMENTS

One of the notable features of Amazon's 2007 Dynamo paper was the use of vector clocks for conflict resolution.

However, the two most prominent systems designed by engineers who worked on Dynamo -- Cassandra and DynamoDB -- both avoid the use of vector clocks in favor of finer-grained updates. To understand why, let's first look at the problem that vector clocks solve.

Resolving conflicts with vector clocks

The original Dynamo, like the open-source Voldemort and Riak, was a key/value database. Thus, objects would need to be serialized in a format such as json. For example, I might have a user object with key `jbellis` and value of `{'email': 'jbellis@example.com', 'phone': '555-5555'}`. We'll call this initial value V0.

Next, suppose we update the email address, changing the value to V1 of `{'email': 'jbellis@illustration.com', 'phone': '555-5555'}`. Some failure causes this to only be written to one replica. Later, we update the phone number, but we read from a different replica so we start from the original value V0 (with the original email address), so we write V2 `{'email': 'jbellis@example.com', 'phone': '444-4444'}`.

(Note that failure -- whether actual machine failure, network failure, or even load shedding -- can cause "conflicting" updates even with a single client and no concurrency.)

Since our object values are opaque blobs to this system, a naive last-write-wins conflict resolution policy will result in discarding the V1 email address change in favor of the V2 phone number update. This is why it's so easy to lose data using last-write-wins conflict resolution in a key/value system like Riak.

Vector clocks solve this problem by allowing the database to push conflict resolution back out to the client. Skipping a lot of details, the database would retain both V1 and V2, and when a client next reads key `jbellis`, it would return both versions and tell the client, "you figure out what you want the value to be now." The client can then deserialize the objects and merge the separately updated fields without data loss to the intended value of `{'email': 'jbellis@illustration.com', 'phone': '444-4444'}`.

Problems with vector clocks

I see three main problems with a key/value database like Dynamo and its first-generation open-source derivatives:

1. Performance: as I alluded to earlier this year, updating a single field in an object stored in a key/value database requires three steps: read and deserialize the exiting object, update the desired field, and serialize and write the resulting object as a new value. Updating an object in Cassandra requires only communicating the changed fields, no more.

Riak both had to go beyond vector clocks when implementing counters.

Cassandra's solution

People who have been burned by last-write-wins in other systems are justifiably nervous when approaching Cassandra. But Cassandra breaks a row up into columns that can be updated independently. Here's what that looks like for our example:

```
CREATE TABLE users (
    username text PRIMARY KEY,
    email text,
    phone text
);

INSERT INTO users (username, email, phone)
VALUES ('jbellis', 'jbellis@example.com', '555-5555');

UPDATE users SET email = 'jbellis@illustration.com' WHERE username = 'jbellis';

UPDATE users SET phone = '444-4444' WHERE username = 'jbellis';
```

This way, the storage engine can resolve changes to email and phone columns automatically. Conversely, if there are concurrent changes to a single field, only one will be retained, which is also what we want. (Cassandra extends this fine-grained conflict resolution to Collection elements as well.)

Thus, clock synchronization is nice to have in a Cassandra cluster but not critical; timestamps are only used to pick a "winning" update within a single column or collection element. (A timestamp tie will also result in a deterministic, commutative result.) Lightweight transactions are available when linearizability is important.

What Cassandra gives up here is the ability to create custom behavior for updates-based-on-existing-values. However, as counters illustrate, vector clocks are often inadequate for this as well. Usually you end up having to store the entire version history, which Cassandra supports by clustering with uuids.

Summary

Cassandra addresses the problem that vector clocks were designed to solve by breaking up documents/objects/rows into units of data that can be updated and merged independently. This allows Cassandra to offer improved performance and simpler application design.

Comments

1.  *zznate* says:
   September 4, 2013 at 11:03 am
   http://www.youtube.com/watch?v=sJJzDB9RhzA

   See ~ 44:09 to see Jonathan breaking this topic down on the whiteboard.

   Reply
2.  *Jonathan Ellis* says:
   September 4, 2013 at 12:17 pm
   For the record, I do not feel that I did a very good job explaining that on the spur of the moment. This post is an attempt to do better.

   Reply
3.  *Sean Cribbs* says:
   September 4, 2013 at 1:07 pm
   Way to misrepresent vector clock usage in Riak! LWW deliberately ignores the vector clock. No one would use that in production without a strong assurance that they will never have concurrent writes. Also note that later in the post Kyle shows how using them properly leads to zero data-loss.

   Reply
4.  *Jerdavis* says:
   September 4, 2013 at 2:17 pm
   What always made me nervous about using a timestamp is: What happens when the system clock gets foo'd up? I'm not talking about some pathological race condition between two updates, but a more common clock drift or (often clock hardware malfunction) or SNTP issue. It seems like all of our hard work to preserve data integrity in the face of failure is susceptible to a single point of failure in the clock source.

September 4, 2013 at 2:32 pm

But it's been a while since I used Cassandra, and there weren't transactions,etc back then.

Reply

5.  *Jonathan Ellis* says:

September 4, 2013 at 2:22 pm

Sean, I think you're looking for malice where none was intended. I acknowledge early in this post that VC solves the problem it's meant to solve; my intent is to communicate (1) why LWW at the column level — where, by definition, you only WANT the most recent value! — is fundamentally different from LWW at the document level and (2) some of the drawbacks of VC that made Cassandra and DynamoDB avoid them.

Am I incorrect that LWW is the Riak default?

Reply

6.  *aphyr* says:

September 4, 2013 at 2:33 pm

"why LWW at the column level — where, by definition, you only WANT the most recent value!"

I disagree that LWW is the only desirable causality type for cells; this makes it impossible to write a value which is guaranteed to be causally connected to a later state of the system. The only circumstances under which you *can* mutate cells safely with LWW require strong coordination of timestamps from an external resolution system, or linearizable transactions like the new Paxos operations.

Reply

7.  *Jonathan Ellis* says:

September 4, 2013 at 3:09 pm

I would agree with that. My point is that rather than optimizing for that situation (casually connected cells) with vector clocks, Cassandra provides a set of tools (LWW-per-cell, UUIDs when you need the full history, and LWT for linearizability) that allow you to solve the same problems without the complexity (server-side and client-side) that VC brings to the table, and with substantially better performance.

8.  *jsmorph* says:

   September 4, 2013 at 4:18 pm

   Jonathan, your last comment (3:09) is a good clarification. Once inside a column, Cassandra offers LWW, UUIDs, and LWT. The latter, of course, is complex. Paxos ("four round trips" in your other post) versus vector clocks is an interesting topic. So I think the post title is a bit misleading. Within a column, Cassandra can (as of 2.0, I guess) use Paxos instead of vector clocks.

   Reply

9.  *Pavel* says:

   March 1, 2016 at 8:40 am

   This link "breaks a row up into columns" seems broken

   Reply

10.  *Joshua Muzaaya* says:

    January 16, 2017 at 9:04 pm

    This article misrepresents Riak as product. Riak doesnot throw away revisions of the same object based on last-write wins. For conflict resolution, Riak presents to you all copies of the modified object during a split and u will decide which one is the most recent or reasonable based on your application.

    Reply

Comments

Your email address will not be published. Required fields are marked *

Leave a comment...

Name *

Email *

Email Address

SUBMIT

« What's New in Cassandra 2.0: Prototype Triggers Support

DataStax OpsCenter 3.2.2 Available »