Occasionally useful posts about RIAs, Web scale computing & miscellanea

HBase vs Cassandra: why we moved

with 43 comments

My team is currently working on a brand new product – the forthcoming MMO www.FightMyMonster.com. This has given us the luxury of building against a NOSQL database, which means we can put the horrors of MySQL sharding and expensive scalability behind us. Recently a few people have been asking why we seem to have changed our preference from HBase to Cassandra. I can confirm the change is true and that we have in fact almost completed porting our code to Cassandra, and here I will seek to provide an explanation.

For those that are new to NOSQL, in a following post I will write about why I think we will see a seismic shift from SQL to NOSQL over the coming years, which will be just as important as the move to cloud computing. That post will also seek to explain why I think NOSQL might be the right choice for your company. But for now I will simply relay the reasons why we have chosen Cassandra as our NOSQL solution.

Caveat Emptor – if you're looking for a shortcut to engaging your neurons be aware this isn't an exhaustive critical comparison, it just summarizes the logic of just another startup in a hurry with limited time and resources!!

Did Cassandra's bloodline foretell the future?

One of my favourite tuppences for engineers struggling to find a bug is "breadth first not depth first". This can be annoying for someone working through complex technical details, because it implies that the solution is actually much simpler if they only looked (advice: only use this saying with established colleagues who will forgive you). I coined this saying because in software matters I find that if we force ourselves to examine the top level considerations first, before tunnelling down into the detail of a particular line of enquiry, we can save enormous time.

So before getting technical, I'll mention I might have heeded my motto better when we were making our initial choice between HBase and Cassandra. The technical conclusions behind our eventual switch might have been predicted: HBase and Cassandra have dramatically different bloodlines and genes, and I think this influenced their applicability within our business.

Loosely speaking, HBase and its required supporting systems are derived from what is known of the original Google BigTable and Google File System designs (as known from the Google File System paper Google published in 2003, and the BigTable paper published in 2006). Cassandra on the other hand is a recent open source fork of a standalone database system initially coded by Facebook, which while implementing the BigTable data model, uses a system inspired by Amazon's Dynamo for storing data (in fact much of the initial development work on Cassandra was performed by two Dynamo engineers recruited to Facebook from Amazon).

In my opinion, these differing histories have resulted in HBase being more suitable for data warehousing, and large scale data processing and analysis (for example, such as that involved when indexing the Web) and Cassandra being more suitable for real time transaction processing and the serving of interactive data. Writing a proper study of that hypothesis is well beyond this post, but I believe you will be able to detect this theme recurring when considering the databases.

NOTE: if you are looking for lightweight validation you'll find the current makeup of the key committers interesting: the primary committers to HBase work for Bing (M$ bought their search company last year, and gave them permission to continue submitting open source code after a couple of months). By contrast the primary committers on Cassandra work for Rackspace, which supports the idea of an advanced general purpose NOSQL solution being freely available to counter the threat of companies becoming locked in to the proprietary NOSQL solutions offered by the likes of Google, Yahoo and Amazon EC2.

Malcolm Gladwell would say my unconscious brain would have known immediately that my business would eventually prefer Cassandra based upon these differing backgrounds. It is horses for courses. But of course, justifying a business decision made in the blink of an eye is difficult…

Which NOSQL database has the most momentum?

Another consideration that has persuaded us to move to Cassandra is a belief that it is now has the most general momentum in our community. As you know, in the business of software platforms the bigger you get the bigger you get – where platforms are perceived as similar, people tend to aggregate around the platform that is going to offer the best supporting ecosystem in the long term (i.e. where the most supporting software is available from the community, and where the most developers are available for hire). This effect is self-reinforcing.

When starting with HBase, my impression then was that it had the greatest community momentum behind it, but I now believe that Cassandra is coming through much stronger. The original impression was partly created by two very persuasive and excellently delivered presentations given by the CTOs of StumpleUpon and Streamy, two big players in the Web industry who committed to HBase some time before Cassandra was really an option, and also from a quick reading of an article entitled "HBase vs Cassandra: NoSQL Battle!" (much of which has now been widely debunked).

Proving momentum comprehensively is difficult to do, and you will have to poke about for yourself, but one simple pointer I offer you is the developer activity on IRC. If you connect to freenode.org and compare the #hbase and #cassandra developer channels, you will find Cassandra typically has twice the number of developers online at any time.

If you consider Cassandra has been around for half as long as HBase, you can see why this is quite a clear indication of the accelerating momentum behind Cassandra. You might also take note of the big names coming on board, such as Twitter, where they plan broad usage (see here).

Note: Cassandra's supporting website looks much lovelier than HBase's, but seriously, this could be a trend driven by more than the marketing. Read on!

Deep down and technical: CAP and the myth of CA vs AP

There is a very powerful theorem that applies to the development of distributed systems (and here we are talking about distributed databases, as I'm sure you've noticed). This is known as the CAP Theorem, and was developed by Professor Eric Brewer, Co-founder and Chief Scientist of Inktomi.

The theorem states, that a distributed (or "shared data") system design, can offer at most two out of three desirable properties – Consistency, Availability and tolerance to network Partitions. Very

basically, “consistency” means that if someone writes a value to a database, thereafter other users will immediately be able to read the same value back, “availability” means that if some number of nodes fail in your cluster the distributed system can remain operational, and “tolerance to partitions” means that if the nodes in your cluster are divided into two groups that can no longer communicate by a network failure, again the system remains operational.

Professor Brewer is an eminent man and many developers, including many in the HBase community, have taken it to heart that their systems can only support two of these properties and have accordingly worked to this design principle. Indeed, if you search online posts related to HBase and Cassandra comparisons, you will regularly find the HBase community explaining that they have chosen CP, while Cassandra has chosen AP – no doubt mindful of the fact that most developers need consistency (the C) at some level.

However I need to draw to your attention to the fact that these claims are based on a complete non sequitur. The CAP theorem only applies to a single distributed algorithm (and here I hope Professor Brewer would agree). But there is no reason why you cannot design a single system where for any given operation, the underlying algorithm and thus the trade-off achieved is selectable. Thus while it is true that a system may only offer two of these properties per operation, what has been widely missed is that a system can be designed that allows a caller to choose which properties they want when any given operation is performed. Not only that, reality is not nearly so black and white, and it is possible to offer differing degrees of balance between consistency, availability and tolerance to partition. This is Cassandra.

This is such an important point I will reiterate: the beauty of Cassandra is that you can choose the trade-offs you want on a case by case basis such that they best match the requirements of the particular operation you are performing. Cassandra proves you can go beyond the popular interpretation of the CAP Theorem and the world keeps on spinning!

For example, let’s look at two different extremes. Let us say that I must read a value from the database with very high consistency – that is, where I will be 100% sure to receive the last copy of that data which was previously written. In this case, I can read the value from Cassandra specifying consistency level “ALL”, which requires that all the nodes that hold replicated copies of that data agree on its value. In this case, I have zero tolerance to either node failure, or network partition. At the other extreme, if I do not care about consistency particularly, and simply want the maximum possible performance, I can read the value from Cassandra using consistency level “ONE”. In this case, a copy is simply taken from a random node amongst those holding the replicas – and in this case, if the data is replicated three times, it does not matter if either of the two other nodes holding copies have failed or been partitioned from us, although now of course it is also possible that such conditions may mean the data I read is stale.

And better still, you are not forced to live in a black and white world. For example, in our particular application important read/write operations typically use consistency level “QUORUM”, which basically means – and I simplify so please research before writing your Cassandra app – that a majority of nodes in the replication factor agree. From our perspective, this provides both a reasonable degree of resilience to node failure and network partition, while still delivering an extremely high level of consistency. In the general case, we typically use the aforementioned consistency level of “ONE”, which provides maximum performance. Nice!

For us this is a very big plus for Cassandra. Not only can we now easily tune our system, we can also

design it so that, for example, when a certain number of nodes fail, or the network connecting those nodes falters, our service continues operating in many respects, and only those aspects that require data consistency fail. HBase is not nearly so flexible, and the pursuit of a single approach within the system (CP) reminds me of the wall that exists between SQL developers and the query optimizer – something it is good to get beyond!

In our project then, Cassandra has proven by far the most flexible system, although you may find your brain at first loses consistency when considering your QUORUMs.

When is monolithic better than modular?

An important distinction between Cassandra and HBase, is that while Cassandra comes as a single Java process to be run per node, a complete HBase solution is really comprised of several parts: you have the database process itself, which may run in several modes, a properly configured and operational hadoop HDFS distributed file system setup, and a Zookeeper system to coordinate the different HBase processes. Does this mean then that this is a modularity win for HBase?

Although it is true that such a setup might promise to leverage the collective benefits of different development teams, in terms of systems administration the modularity of HBase cannot be considered a plus. In fact, especially for a smaller startup company, the modularity of HBase might be a big negative. Let me explain…

The underpinnings of HBase are pretty complex, and anyone in doubt of this should read the original Google File System and BigTable papers. Even setting up HBase in pseudo distributed mode on a single server is difficult – so difficult in fact that I did my best to write a guide that takes you past all the various gotchas in the minimum time (see http://ria101.wordpress.com/2010/01/28/setup-hbase-in-pseudo-distributed-mode-and-connect-java-client/ if you wish to try it). As you will see from that guide, getting HBase up and running in this mode actually involves setting up two different system systems manually: first hadoop HDFS, then HBase itself.

Now to the point: the HBase configuration files are monsters, and your setup is vulnerable to the quirks in default network configurations (in which I include both the default networking setups on Ubuntu boxes, and the subtleties of Elastic IPs and internally assigned domain names on EC2). When things go wrong, you will be presented with reams of output in the log file. All the information you need to fix things is in there, and if you are a skilled admin you are going to get through it.

But what happens if it does wrong in production and you need to fix it in a hurry? And what happens if like us, you have a small team of developers with big ambitions and can't afford a team of crack admins to be on standby 247?

Look seriously, if you're an advanced db admin wanting to learn a NOSQL system, choose HBase. It's so damn complex that safe pairs of hands are going to get paid well.

But if you're a small team just trying to get to the end of the tunnel like us, wait 'til you hear the Gossip…

It's Gossip talk dude, Gossip!

Cassandra is a completely symmetric system. That is to say, there are no master nodes or region servers

like in HBase – every node plays a completely equal role in the system. Rather than any particular node or entity taking on a coordination role, the nodes in your cluster coordinate their activities using a pure P2P communication protocol called “Gossip”.

A description of Gossip and the model using it is beyond this post, but the application of P2P communication within Cassandra has been mathematically modelled to show that, for example, the time taken for the detection of node failure to be propagated across the system, or for a client request to be routed to the node(s) holding the data, occur deterministically within well bounded timeframes that are surprisingly small. Personally I believe that Cassandra represents one of the most exciting uses of P2P technology to date, but of course this idea is not relevant to choosing your NOSQL database!

What is relevant are the real benefits that the Gossip-based architecture gives to Cassandra’s users. Firstly, continuing with the theme of systems administration, life becomes much simpler. For example, adding a new node to the system becomes as simple as bootstrapping its Cassandra process and pointing it at a seed node (an existing node within your cluster). When you think of the underlying complexity of a distributed database running across, potentially, hundreds of nodes, the ability to add new nodes to scale up with such ease is incredible. Furthermore, when things go wrong you no longer have to consider what kind of nodes you are dealing with – everything is the same, which can make debugging a more progressive and repeatable process.

Secondly I have come to the conclusion that Cassandra’s P2P architecture provides it with performance and availability advantages. Load can be very evenly balanced across system nodes thus maximizing the potential for parallelism, the ability to continue seamlessly in the face of network partitions or node failures is greatly increased, and the symmetry between nodes prevents the temporary instabilities in performance that have been reported with HBase when nodes are added and removed (Cassandra boots quickly, and its performance scales smoothly as new nodes are added).

If you are looking for more evidence, you will be interested to read a report from a team with a vested interest in hadoop (i.e. which should favor HBase)…

A report is worth a thousand words. I mean graph right?

The first comprehensive benchmarking of NOSQL systems performed by Yahoo! Research now seems to bear out the general performance advantage that Cassandra enjoys, and on the face of it the figures do currently look very good for Cassandra.

At the time of writing these papers are in draft form and you can check them out here:
http://www.brianfrankcooper.net/pubs/ycsb-v4.pdf
http://www.brianfrankcooper.net/pubs/ycsb.pdf

NOTE: in this report HBase performs better than Cassandra only respect of range scans over records. Although the Cassandra team believes they will quickly approach the HBase times, it is also worth pointing out that in a common configuration of Cassandra range scans aren’t even possible. I recommend this to you as being of no matter, because actually in practice you should implement your indexes on top of Cassandra, rather than seek to use range scans. If you are interested in issues relating to range scans and storing indexes in Cassandra, see my post here http://ria101.wordpress.com/2010/02/22/cassandra-randompartitioner-vs-orderpreservingpartitioner/).

FINAL POINT OF INTEREST: the Yahoo! Research team behind this paper are trying to get their

benchmarking application past their legal department and make it available to the community. If they succeed, and I hope they do, we will be treated to an ongoing speed competition galore, and both HBase and Cassandra will doubtless be improving their times further.

A word on locking, and useful modularity

You may no doubt hear from the HBase camp that their more complex architecture is able to give you things that Cassandra's P2P architecture can't. An example that may be raised is the fact that HBase provides the developer with row locking facilities whereas Cassandra cannot (in HBase row locking can be controlled by a region server since data replication occurs within the hadoop layer below, whereas in Cassandra's P2P architecture all nodes are equal, and therefore none can act as a gateway that takes responsibility for locking replicated data).

However, I would reflect this back as an argument about modularity, which actually favours Cassandra. Cassandra implements the BigTable data model but uses a design where data storage is distributed over symmetric nodes. It does that, and that's all, but in the most flexible and performant manner possible. But if you need locking, transactions or any other functionality then that can be added to your system in a modular manner – for example we have found scalable locking quite simple to add to our application using Zookeeper and its associated recipes (and other systems such as Hazelcast might also exist for these purposes, although we have not explored them).

By minimizing its function to a narrower purpose, it seems to me that Cassandra manages to implement a design that executes that purpose better – as indicated for example by its selectable CAP tradeoffs. This modularity means you can build a system as you need it – want locking, grab yourself Zookeeper, want to store a full text index, grab yourself Lucandra, and so on. For developers like us, this means we don't have to take on board more complexity than we actually need, and ultimately provides us with a more flexible route to building the application we want.

MapReduce, don't mention MapReduce!

One thing Cassandra can't do well yet is MapReduce! For those not versed in this technology, it is a system for the parallel processing of vast amounts of data, such as the extraction of statistics from millions of pages that have been downloaded from the Web. MapReduce and related systems such as Pig and Hive work well with HBase because it uses hadoop HDFS to store its data, which is the platform these systems were primarily designed to work with. If you need to do that kind of data crunching and analysis, HBase may currently be your best option.

Remember, it's horses for courses!

Therefore as I finish off my impassioned extolation of Cassandra's relative virtues, I should point out HBase and Cassandra should not necessarily be viewed as out and out competitors. While it is true that they may often be used for the same purpose, in much the same way as MySQL and Postgres, what I believe will likely emerge is that they will become preferred solutions for different applications. For example, as I understand StumbleUpon has been using HBase with the associated hadoop MapReduce technologies to crunch the vast amounts of data added to its service. Twitter is now using Cassandra for real time interactive community posts. Our needs fit better with the interactive serving and processing of data and so we are using Cassandra, and probably to some degree there you have it.

As a controversial parting shot though the gloves are off for the next point!

NOTE: before I continue I should point out Cassandra has hadoop support in 0.6, so its MapReduce integration may be about to get a whole load better.

O boy, I can't afford to lose that data…

Perhaps as a result of the early CAP Theorem debates, an impression has grown that data is somehow safer in HBase than Cassandra. This is a final myth that I wish to debunk: in Cassandra, when you write new data it is actually immediately written to the commit log on one of the nodes in the quorum that will hold the replicas, as well as being replicated across the memory of the nodes. This means that if you have a complete power failure across your cluster, you will likely lose little data. Furthermore once in the system, data entropy is prevented using Merkle trees, which further add to the security of your data :)

In truth I am not clear exactly what the situation with HBase is – and I will endeavour to update this post as soon as possible with details – but my current understanding is that because hadoop does not yet support append, HBase cannot efficiently regularly flush its modified blocks of data to HDFS (whereupon the new mutations to data will be replicated and persisted). This means that there is a much larger window where your latest changes are vulnerable (if I am wrong, as I may be, please tell me and I will update the post).

So while the Cassandra of Greek mythology had a rather terrible time, the data inside your Cassandra shouldn't.

NOTE: Wade Arnold points out below that (at the time of writing this) hadoop .21 is about to be released, which will solve this problem with HBase.

Possibly related posts: (automatically generated)

* HBase vs Cassandra & comparing NOSQL solutions
* HBase, newbie install and configuration tips
* Cassandra: RandomPartitioner vs OrderPreservingPartitioner
* SQL or NoSQL?

Written by dominicwilliams

February 24, 2010 at 7:27 pm

Posted in Cassandra, HBase, NOSQL Databases, hadoop

Tagged with Cassandra, evaluate, HBase, hbase vs cassandra, performance
« Cassandra: RandomPartitioner vs OrderPreservingPartitioner
Red5 cabin fever – advanced scope and room management »
43 Responses

Subscribe to comments with RSS.

1.

Cassandra 0.6 (in beta now) supports Hadoop map/reduce, btw.

Jonathan Ellis

February 24, 2010 at 8:54 pm
Reply

*

Thanks, updated post with note.

dominicwilliams

February 24, 2010 at 9:27 pm
Reply

2.

Your HDFS append comment is correct. This will require hadoop .21 which is currently being voted on with an imminent release as all tests are passing.

Wade Arnold

February 24, 2010 at 9:25 pm
Reply

*

Thanks Wade, I've appended a note to the article drawing attention to this.

dominicwilliams

February 25, 2010 at 6:38 pm
Reply

*

I don't know how true that is… I really want the append/fsync feature in HDFS 0.21 but it appears that the 0.21 release process has stalled out. Unless you have more recent info. This thread seems to show no conclusion…

http://mail-archives.apache.org/mod_mbox/hadoop-general/201002.mbox/

jerdavis

March 19, 2010 at 4:47 pm
Reply

3.

I think it's so cool when people abbreviate Microsoft with a dollar sign (M$) . They seem so uber-1337 and take such a stand.

Softie

February 24, 2010 at 10:24 pm
Reply
*

You're probably right its a bit pathetic. The trouble is they've had so much of my money and I'm still getting over it.

dominicwilliams

February 24, 2010 at 10:47 pm
Reply

4.

[...] February 24, 2010 at 6:02 pm · Filed under Cloud computing ·Tagged Databases [From HBase vs Cassandra: why we moved « Bits and Bytes.] [...]

HBase vs Cassandra: why we moved « Bits and Bytes. « The other side of the firewall

February 24, 2010 at 11:02 pm
Reply

5.

interesting read — the very first 1-2 paragraphs almost turned me off however….. anyways I kept reading and it definitely kept my interest — as stated I think comparing cassandra to hbase and friends is like comparing apples and oranges…. we've been looking at big data for a while for one of our projects — nice to see the steam building up around the web

ian

February 25, 2010 at 2:04 am
Reply

6.

CouchDB! It offers a robust map/reduce implementation, replication, and it talks http and json natively. It's incredible.

Mark

February 25, 2010 at 5:26 am
Reply

7.

As Wade says, the Hadoop team works very closely with the HBase team to get the last issue with HDFS append resolved for the next release. Here is my take on HBase's write-ahed-log: http://www.larsgeorge.com/2010/01/hbase-architecture-101-write-ahead-log.html

One thing that sort of was never answered is scalability, while both systems are designed to do so it is known from the Dynamo paper that the Gossip protocol may eventually saturate the network and is

something that needs to be improved on. Is that “fixed” in Cassandra?

With BigTable you get a virtually boundless scalability with linear scan times and logarithmic read performance, not matter the size of the cluster. So with HBase the use-case I see is to be able to create a huge storage system and be able to efficiently process the data. Cassandra seems to be easier for rapid prototyping and flexibility along the way to determine what is needed for an otherwise unknown use-case. HBase has indeed a more strict model compared to that but is also on the other hand not promising the world. Bottom line is as you say, “horses for courses” and that this is no and should never be a shootout between NoSQL systems.

As far as community is concerned, this is to be expected. A system backed by trendy sites like Facebook and Twitter ought to attract more developers while the rather more involved stup for a basic HBase cluster plus the fact that “M$” now uses it is just less spiffy. To me this is no indicator for success but for hype. And hype must not necessarily be a gauge for what is “better or worse”.

larsgeorge

February 25, 2010 at 8:31 am
Reply
*

FB, Twitter, etc. are using Cassandra because it is a better technical choice, not to be “trendy.” Ryan King listed some of their criteria for choosing Cassandra over at http://nosql.mypopescu.com/post/407159447/cassandra-twitter-an-interview-with-ryan-king#comment-36088396 .

Jonathan Ellis

February 25, 2010 at 7:17 pm
Reply
o

Of course, I was just referring to the fact that there are more people on IRC for Cassandra as opposed to HBase’s channel. Not why they chose it.

larsgeorge

February 25, 2010 at 7:36 pm
o

Ah. Context is important there. Remember, in Bradford’s anti-Cassandra FUD piece, he claims that one of the reasons you should use HBase is that it’s more popular. So whether it is a useful metric or not, it’s worth correcting the record. :)

Jonathan Ellis

February 25, 2010 at 7:40 pm
8.

This was a great reading indeed. Thanks for taking the time :)

Marc

February 25, 2010 at 9:24 am
Reply

9.

Interesting insights. Thanks for the in-depth post.

Andrew

February 25, 2010 at 5:57 pm
Reply

10.

[...] HBase vs Cassandra: why we moved My team is currently working on a brand new product – the forthcoming MMO http://www.FightMyMonster.com. This has given us [...] [...]

Top Posts — WordPress.com

February 26, 2010 at 12:36 am
Reply

11.

Thanks Dominic for sharing your thoughts.

I'm at a small startup and we've been happily using HBase in production on Amazon EC2 for 7 months. We have only 2 engineers who are primarily developers rather than db admins, and I can relate to your experience with the complexity of getting started. You have to learn how to work with HDFS, Zookeeper as well as HBase and there are so many configuration options.

We also have the fear you talk about: what will we do if we run into a serious low-level production problem. There don't seem to be any companies offering HBase support contracts. However, the community support has been great. We've always received quick responses on the HBase Users mailing list and IRC channel. The HBase committers patiently guided us through the one or two production problems we did encounter.

Anyway, your post and the recent news from Twitter has definitely made me more curious about Cassandra. So much to learn in this space…

Ken Weiner

February 26, 2010 at 7:27 pm
Reply

*

Hi Ken,

If you do decide you require HBase support, I'm happy to discuss the support levels we provide at Cloudera. Let me know.

Thanks,
Kylie

Kylie Clement

March 4, 2010 at 1:38 am
Reply

12.

Great comparison thanks Dom. In the spirit of cloud it'll be interesting to see how services can remove even more of the admin work. Take SimpleDB and their recent "consistent read" announcement or the comment above re: hbase on EC2….

Very cool.

Chad Arimura

February 27, 2010 at 6:39 pm
Reply

13.

nice post!

just one thing that I'd like to mention about the "C" in Cassandra's CAP. even when consistency level is set to ALL this doesn't guarantee does it?

consider the case when 2 clients perform a WRITE operation (WRITE1 & WRITE2), at the same time, to the same data, with consistency level ALL. WRITE1 may arrive first at half of the replicas while WRITE2 may arrive first at the other half.
if these writes have no logic order there will be no way for the system to know which occurred "first", and the decision will be left to the clients on future reads.
I realize this is in the nature of eventually consistent systems, my point is only that people should be made aware of Cassandra's/Riak's/Dynamo's consistency limitations

On the other hand… I may be wrong too. But if you agree, then I think its a valuable piece of information to include in your post

Alex

alex

February 28, 2010 at 6:28 pm
Reply

*

Cassandra uses client-supplied timestamps to resolve conflicts like the one you describe.

Jonathan Ellis

March 1, 2010 at 2:10 pm
Reply
- o

Yeah, that's what I was getting at.

From my understanding, even with consistency level set to ALL you are not guaranteed that all replicas will have a consistent view of the world.

Do I have that right?

My point (if the above is not completely wrong) is that Cassandra NEVER promises to be strongly consistency, regardless of settings.

alex

March 1, 2010 at 2:40 pm
- o

In the improbable case where the client-supplied time stamps are of:
(1) "long" type and are identical
(2) "vector clock" type and have no causal order

Does a client resolve the inconsistency at some point in the future when they perform a read?
Or is the inconsistency resolved deterministically at server side?

alex

March 1, 2010 at 4:26 pm
- o

1) on the first read the server will pick one version and write it back to any replicas that have another version w/ the same ts during read repair

2) cassandra does not yet support vector clocks, but one of the points of adding them is so you can push this kind of conflict back to the client and say "you know more about your domain than I do, you tell me what you want to do here"

Jonathan Ellis

March 1, 2010 at 4:28 pm
- *

Hi Alex, the point is, that if you use consistency level ALL then Cassandra ***isn't*** an "eventually consistent" system i.e. Cassandra does not behave like Dynamo in that case.

If you use ALL for reads and writes on a value, then once you have written that value, subsequent reads will always return the same value (or fail, if a node in the replication factor is unavailable). The reason is that the operation contacts all nodes in the replication factor to ensure consistency.

But note in practice you will get similar results using QUORUM, in which a simple majority of nodes in the replication factor are contacted. This provides better performance, and allows some nodes in the replication factor to be unavailable.

In your Cassandra application, you will vary your consistency level depending upon the operation you are performing. Where consistency is not important, and you are emphasising performance instead, you will use the lowest consistency level of 1.

dominicwilliams

March 1, 2010 at 2:13 pm
Reply
o

Hi Dominic, it seems like either we're talking about different things or one of us is incorrect… very possibly me!

As far as I understand, the scenario I described (2 writers, consistency level = ALL, half receive write1 1st, other half receive write2 1st) would NOT result in all replicas having consistent values, even in the case when there are zero failures.

Just to clarify… I'm not inferring that Cassandra is flawed, I'm just trying to point out what seems to be a common misconception

alex

March 1, 2010 at 3:13 pm
*

Hi Alex, hopefully this is a better answer. Read it too quickly last time..

Firsly, in NOSQL systems reads and writes are performed in separate operations. Thus, even if you always read back the value that you have written (i.e. because you are using ALL or QUORUM) without further measures a lost update can occur. For example two clients might read a number simultaneously, add differing amounts to it, then write it back. When the clients write the value back, the update of at least one client will be lost i.e. the number will only have been incremented by the amount added by the "winning" client.

By contrast, with an SQL system you can read and write in a single statement, and thus the database systems themselves can serialize these client operations preventing lost updates happening (although in practice, as applications become complex, it is often not possible to encapsulate each operation into a single SQL statement and thus these problems can still occur).

In order to avoid lost updates and related problems with NOSQL systems (and sometimes as mentioned with SQL systems, and certainly when you are sharding SQL systems) you have to use some kind of global synchronization mechanism. We are using ZooKeeper for this, which comes with Java recipes you can can use to create scalable global write locks and other things. Hazelcast is another system Java developers can look at.

When I talk about a system being "eventually consistent", I mean that it has the characteristic that you can write a value for some data, and then read it back with another value. This is the case for example with Amazon EC2's SimpleDb (or at least it was until a few days ago, since they have just released new features allowing caller to specify desired consistency level).

If you have this kind of "eventual consistency", you cannot use global synchronization to be sure you are performing operations like debiting one bank account and crediting another correctly. However so long as you can always read back the value you just wrote, you can use global synchronization to do this (note that you can also do transactions with ZooKeeper, but since in our game the accounts we deal with don't contain real money, we haven't gone that far :))

Personally, I think vector clocks will be a useful addition for high performance or high availability scenarios where you don't want to enforce global serialization (or maybe even use strong consistency) but still need to be able to resolve conflicts in data in the rare case where they occur. However I reckon for most people global serialization will be the preferred option if conflicts must be dealt with simply because writing client-side conflict resolution code is always going to drag on productivity.

dominicwilliams

March 1, 2010 at 5:19 pm
Reply

14.

Zeitgeist at work: Amazon CTO Werner Vogels on the same wavelength?

Just 3.5 hours after this post was made (after timezone differences), Werner Vogels made a short post covering similar ground and announcing the introduction of configurable consistency features to SimpleDb.

This post was widely tweeted within the NOSQL community within minutes of being posted, and included two key subjects:

1. The advantages of the configurable consistency model (which at the time of writing, only Cassandra provided).

2. A discussion of Eric Brewer's CAP Theorem in the context of NOSQL (although this post also focuses on how this theorem has been widely misunderstood in relation to NOSQL and other shared data systems – Eric Brewer sometimes describes the CAP Theorem as applying to "shared data systems" in general, e.g. see http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf, which mislead a lot of people. In fact, the CAP Theorem can only possibly be applied to a specific set of algorithms that are used to minimally implement a shared data system, and there is no reason why you cannot design a system that makes the algorithms and thus tradeoffs configurable on an operation

by operation basis, as does Cassandra).

No doubt EC2 was planning to add configurable consistency to SimpleDb anyway, but maybe this post and the buzz it was getting bumped Werner Vogels into announcing it?

Who knows, but it’s a nice coincidence!

Check out his post at
http://www.allthingsdistributed.com/2010/02/strong_consistency_simpledb.html

dominicwilliams

March 1, 2010 at 6:54 pm
Reply

15.

[...] Getting Real about NoSQL and the SQL-Isn’t-Scalable Lie. В продолжение темы: HBase vs Cassandra: why we moved и Brewer’s CAP [...]

Интересные ссылки №205 - max - блог разработчиков

March 5, 2010 at 7:05 am
Reply

16.

Thank you for this, Dominic. It covers a lot of the issues that most people seem to miss when they’re making these same sorts of decisions.

Jeff Darcy

March 19, 2010 at 4:10 pm
Reply

17.

If you need point queries (value by key), short scan queries (lookup all the versions of the same row/record) and efficient large scan implementation for MapReduce jobs in one package than you choice is obvious – HBase. Good write up anyway.

Vladimir Rodionov

March 19, 2010 at 9:16 pm
Reply

18.

[...] http://ria101.wordpress.com/2010/02/24/hbase-vs-cassandra-why-we-moved/ □□□ Dominic Williams □□□□□ February 24, 2010 at 7:27 pm [...]

HBase vs Cassandra: □□□□□□□□□» □□□□

March 24, 2010 at 4:10 pm
Reply

19.

[...] FightMyMonster team switched from HBase to Cassandra after concluding that "HBase is more suitable for data warehousing, and large scale data [...]

Cassandra ライブ □□がテンコ □り ɟonathan Ellis @ Rackspace [#cassandra #nosql] « Agile Cat — Azure & Hadoop — Talking Book

March 26, 2010 at 2:28 am
Reply

20.

Hadoop .21 is a farce with Yahoo proposing to scrapping it and back porting some of its fixes/features to .20 and to begin .22. HDFS Append is still in limbo.

seymourz

March 28, 2010 at 6:06 am
Reply

21.

The descriptions are almostly right, but following tow:
1. The reference of Yahoo's benchmark report is not good.
2. Pithy is better, and save writer and reader's time.

Thanks for you good post.

schubertzhang

March 31, 2010 at 11:15 am
Reply

22.

[...] April 3, 2010 Sam Baskinger Leave a comment Go to comments Dominic Williams has an excellent post at Bits and Bytes on why he (his company, really) moved to Cassandra. This post is a simple [...]

Cassandra Notes from Bits and Bytes « Coffee's Gone Already?

April 4, 2010 at 4:02 am
Reply

23.

[...] HBase vs Cassandra: why we moved [...]

Cheatsheet: 2010 04.01 ~ 04.07 - gOODiDEA.NET

April 7, 2010 at 2:19 am
Reply

24.

Dominic, I'd love to hear more about your experiences since you made the switch… especially around the day-to-day admin of the cluster nodes and the amount of attention they've needed? Have you found that your original design decisions were on the money or do you wish you'd structured things differently?

thanks!

Andrew

April 7, 2010 at 11:25 am
Reply
*

Hi, FightMyMonster.com is not on general release yet, so I won't vouch for its scalability in production – but we've now been working with Cassandra for a while, have ported all our existing code base for FMM over and have also integrated related systems like Lucandra.

What I can report is that in terms of general ease of configuration, setting up clusters and maintaining them, developing against its model etc we are very happy. As compared with our experiences with SQL sharding, HBase etc we are in a much better place, at least for us given our requirements. Based on our experiences, and having convinced ourselves of the NoSQL and in particular Cassandra models, we are planning to base future projects on Cassandra and indeed port some other unrelated ones across.

Bearing in mind the above, below are the only real difficulties we have experienced so far. These are not so much criticisms as a reality check because Cassandra is cutting edge software under constant development:-

1. We access Cassandra from Java. We started off using http://code.google.com/p/cassandra-java-client/ which wrapped the thrift interface (we thought quite nicely). This wasn't tracking Cassandra releases so we migrated to a new client called Hector, which is the one currently most widely in use. The API of this client has changed several times, so there has been quite a bit of refactoring going on. Hector doesn't yet surface some very useful features from the latest Thrift interface like batch mutate – we are considering writing basic connection pool / utility system for working with raw Thrift so keeping up to date easier.

2. Recently had a strange problem where if we deleted a row from database, then created another row with same id, although the operation succeeded we could not see the row. The reason we have concluded was that the rows in question contained super columns and had been originally created with an earlier version of Cassandra that had a bug documented in JIRA 703. This problem does not exist with super column rows created / deleted / recreated since 0.6.

3. If your app does some complex and/or critical multi-row data manipulation like us, you are going to need some kind of centralized locking system for case where race conditions could screw things up. We are using ZooKeeper for this purpose, and I plan to open source our locking and

synchronization primitives library for general use asap. HBase offers single row locking, and multi-row locking is planned, but for a whole load of reasons I believe this is much better handled by ZooKeeper where/if this is needed. Once our library is out, this won't be something people have to worry about because managing a ZooKeeper cluster is also very straightforward.

dominicwilliams

April 7, 2010 at 6:32 pm
Reply
o

Thanks for the details. When is FightMyMonster going live? Also… where in London you guys based?

Andrew

April 7, 2010 at 11:20 pm
o

We are doing development work in Queens Park area, but FMM is actually a Swiss operation. Not clear yet when we will be ready to go live, but product is beginning to look v.good so hopefully soon.