Riak and Cassandra and HBase, oh my!

May 18th, 2010 by deinspanjer

We are marching along in our integration of HBase with the Socorro Crash Stats project, but I wanted to take a minute away from that to talk about a separate project the Metrics team has also been involved with.

Mozilla Labs Test Pilot is a project to experiment and analyze data from real world Firefox users to discover quantifiable ways to improve our user experience. I was very interested and excited about the project because of the care they take to protect the user's privacy. They have a very user focused privacy policy that is easy to read, which always makes me happy. Every step of the way they make sure the user is aware and comfortable with the data they are sending by making it easy for the user to see their data before they submit it and providing the user the choice to submit it or not when the data is ready. The data is always very general in nature, not containing any sensitive information like URLs and it is not associated with any personally identifying information at any time.

In the pre 1.0 releases of Test Pilot, the data that is submitted from the add-on is received by a simple script transforms the data into a flat file that is stored on an NFS server.

We are planning on making a huge drive to ramp up the volume of users and the number of experiments, and that means that this simple storage mechanism will not survive. Here are some of the most important requirements we've hashed out in our planning:

* Expected minimum users: 1 million. Design to accommodate 10 million by the end of the year and have a plan for scaling out to tens of millions. (This is the 1x 10x 100x rule of estimation of which I am a fan)
* Expected amount of data stored per experiment: 1.2 TB
* Expected peak traffic: approximately 75 GB per hour for two 8 hour periods following the conclusion of an experiment window. This two day period will result in collection of approximately 90% of the total data.
* Remain highly available under load
* Provide necessary validation and security constraints to prevent bad data from polluting the experiment or damaging the application
* Provide a flexible and easy-to-use way for data analysts to explore the data. While all of these guys are great with statistics and thinking about data, not all of them have a programming background, so higher-level APIs are a plus.
* Do it fast.

I am a technology nut. I love to research technologies to keep abreast of the state-of-the-art and also potential tools. While I've always been a SQL aficionado, I am also a big fan of the "NoSQL" technologies because I feel there is a great role that they serve.

When I looked at the characteristics of this project, I felt that a key-value or column-store solution was the best fit, so I started digging through my research bookmarks and doing some technology cost/benefit analysis.

Eventually, our team came down to three primary contenders:

* HBase
* Cassandra
* Riak

We recently had a meeting wherein we hashed out a lot of the pros and cons of each of these solutions. I wanted to share that discussion with everyone, not because I was looking forward to being set-upon by the two contenders that I didn't feel were the best fit, but rather for two reasons:

1. Crowd-sourcing — I believe that laying out the thoughts and assumptions in the open is the best way to ensure that we receive the broadest set of feedback from the experts in each of the varying technologies. I further believe that it is better to be aware of the over-looked features and warnings raised by these experts and consider what can be done to mitigate them rather than hiding from them.
2. Sharing of knowledge — Even if it turns out that we didn't get all the answers right or that we didn't come up with the ideal solution, I believe that we asked a lot of good questions here and I believe that listing these questions might help some other team who has to make a similar decision.

So let's get down to the discussion points:

* Scalability — Deliver a solution that can handle the expected starting load and that can easily scale out as that load goes up.
* Elasticity — Because the peak traffic periods are relatively short and the non-peak hours are almost idle, it is important to consider ways to ensure the allocated hardware is not sitting idle, and that you aren't starved for resources during the peak traffic periods.
* Reliability — Stability and high availability is important. It isn't as critical as it might be in certain other projects, but if we were down for several hours during the peak traffic period, the client layer needs to be able to retain the data and resubmit at a later date.
* Storage — Need enough room to store active experiments and also recent experiments that are being analyzed. It is expected that data will become stale over time and can be archived off of the active cluster.
* Analysis — What do we have to put together to provide a friendly system to the analysts?
* Cost — Actual cost of the additional hardware needed to deploy the initial solution and to scale through at least the end of the year.
* Manpower — How much time and effort will it take us to deliver the first critical stage of the project and the subsequent stages? Also consider ongoing maintenance and ownership of the code.
* Security — Because we will be accepting data from an outside, untrusted source, we need to consider what steps are necessary to ensure the health of the system and the privacy of users.
* Extensibility — delivering a platform that can readily evolve to meet the future needs of the project and hopefully other projects as well.
* Disaster Recovery / Migration — If the original system fails to meet the requirements after going live, what options do we have to recover from that situation? If we decide to switch to another technology, how do we move the data?

Now we iterate those points again, but this time we have the points made by the team regarding each of the three solutions being considered:

* Elasticity– Machines can be added as load increases. Machines can be turned off and reconfigured to remove them. There is the ever-present the risk of a bug resulting in a lack of replication or corruption causing data loss. In all three solutions, re-balancing the existing data incurs an additional

load penalty as data is shifted around the cluster. We need to consider how much time and manual administration is required, how much can be automated, how risky rebalancing is, and how long until we begin to see the benefit of the additional nodes.

o HBase

In HBase, the data is split into "regions". The backing data files for regions are stored in HDFS and hence replicated out to multiple nodes in the cluster. Every RegionServer owns a set of regions. Normally, the RegionServer will own regions that exist on the local HDFS DataNode.

If you add a new node, HDFS will begin considering that node for the purposes of replication. When a region file is split, HBase will determine which machines should be the owners of the newly split files. Eventually, the new node will store a reasonable portion of the new and newly split data.

Re-balancing the data involves both re-balancing HDFS and then ensuring that HBase reasses the ownership of regions.

o Cassandra

In Cassandra, nodes claim ranges of data. By default, when a new machine is added, it will receive half of the largest range of data. There are configuration options during node start-up to change that behavior. There are certain configuration requirements to ensure safe and easy balancing, and there is a rebalance command that can perform the work throughout all the data ranges. There is also a monitoring tool that allows you to track the progress of the re-balancing.

o Riak

In Riak, the data is divided into partitions that are distributed among the nodes. When a node is added, the distribution of partition ownership is changed and both old and new data will immediately begin migrating over to the new data.

* Cost — Regardless of solution, we should be able to use commodity server hardware with Linux OS.

o HBase — Because of the heavy peak traffic periods, it is very likely that we would need a dedicated cluster. Otherwise, other projects such as Socorro might be negatively impacted. Also, a scheduled maintenance window would affect both projects instead of just one.

HBase is memory-hungry. Our current nodes are dual quad core hyper-threaded boxes with 4TB of disk and 24 GB of memory. It is unlikely that we would want to go less than that. We would need at least two highly available master nodes, and by the end of the year we'd likely need 12 machines for a single cluster solution.

o Cassandra — Much lighter on the memory requirements, especially if you don't need to keep a lot of data in cache. We would likely want to double the amount of CPU on the four nodes currently allocated to the Test Pilot project. We'd also want to order 8 more machines. To perform analysis with Cassandra, we'll have to leverage our Hadoop cluster.

o Riak — Also much lighter on memory requirements. The existing four nodes (quad core 8 GB) allocated for the project should be enough to kick it off, and we'd expect to add at least two more equivalent machines to that cluster. We'd also set up a second cluster of 6 to 8 less powerful machines for the analysis cluster. Because of the elasticity of Riak, we could temporarily re-purpose N-3 of those machines to the write cluster to accommodate expected peak traffic windows.

* Manpower

o HBase — Need a front-end layer to accept experiment submissions from the client. The fewer changes required for the client, the better. Thrift or a roll-our-own Java are the two most likely options. The application needs to be heavily tested for capacity and stability. Likely two weeks for development and two weeks for testing. Estimate is dependent on the amount of security code, sanity checks, and cluster communication fail-over that has to be implemented. Additional maintenance burden of supporting a separate service.

Schema design needs to be reflected in the front-end code to allow data to be parsed out and stored in the proper column families.

o Cassandra — Mostly the same as HBase. Thrift or Java application hand developed and tested. Schema design to accommodate storage by the front-end.

o Riak — Built in REST server. Already heavily tested and production ready. Minimal schema design and no specific hooking in of the schema to the REST server should be needed.

* Security — We can't expect to hide any sort of handshake protocol or authentication token. If we wanted to require an authentication token, extensive changes would have to be made to the client add-on which would delay the project. SSL doesn't seem to gain us much because we aren't transmitting potentially sensitive data, and it has overhead penalties. Our firewall and proxy/load-balancer layer is our most important line of defense. It should reject URL hacks, unusual payload sizes, and potentially be able to blacklist repeated submissions from the same IP. Ideally, if the payload inspection could communicate IP addresses or payload signatures to blacklist, we'd be pretty well equipped to prevent degradation of the cluster health.

o HBase/Cassandra — We would need the custom built front-end layer to be responsible for inspecting the payload to look for invalid/incomplete data and reject it. This adds to the requirements and implementation time of the custom front-end layer.

o Riak — We can use Webmachine pre-commit hooks to allow inclusion of business logic to perform payload inspection.

* Extensibility — When changes are made to the data stored, all three solutions will potentially require modification of the payload inspection routines and potentially the analysis entry-point to reflect the schema changes.

o HBase — Schema changes involving adding or altering column families require disabling the table. This means a maintenance window. Creation of new tables can be performed on the fly.

o Cassandra — Schema changes require a rolling restart of the nodes.

o Riak — New buckets and schema changes are completely dynamic.

* Data Migration — All three solutions make it pretty easy to replicate, export, or MapReduce data out of the system.

* Disaster Recovery — In all three solutions, it would be best for the client add-on to have enough intelligence to be able to back-off if the cluster load is too high, and to retry submission later if it fails.

o HBase — Custom front-end could incorporate fail-over code to locally spool submissions until cluster is back online. A second cluster would be the most viable DR option.

o Cassandra — Same as HBase

o Riak — Could temporarily reassign the entire reporting cluster to handle incoming submissions. Because there is no custom front-end, if we were unable to make the Riak cluster available for client connections, we would have no buffer in place on the server side to spool submissions.

* Reliability — Small periods of downtime should not be a major issue, especially if the client add-on has retry capability and/or if the front-end layer can spool.

o HBase — Until subsequent versions provide better High Availability options, the Hadoop NameNode and HBase Master are still a single point of failure. Certain types of administration and upgrades require restart of the entire cluster with a maintenance window required to modify the NameNode or HBase Master. Rolling restarts are an option for many types of maintenance, but some HBase experts discourage them.

o Cassandra — No single point of failure. Most configuration changes can be handled via rolling restarts.

o Riak — Same as Cassandra.

* Analysis

o HBase — Can provide a HIVE based interface (possibly with JDBC connectivity). Can provide a simplified MapReduce framework to allow analysts to submit certain types of common, simple jobs.

o Cassandra — Uses Hadoop, answer same as HBase.
o Riak — Map Reduce jobs can be written in JavaScript and submitted through the REST API. A light-weight web interface can be created to allow submission of those jobs.

Based on the evaluation of these discussion points, and also on the availability of some Basho experts to deliver a nearly turn-key solution, we have decided to go with Riak for the implementation of the Test Pilot back-end. While it feels a little odd to be using a technology that is similar in many ways to HBase which we are investing heavily in, I think it is the best choice for us and I actually see several areas that we could potentially consider using Riak for other projects.

If you have any questions, concerns, or clarifications, please feel free to submit them as comments and I will respond or update the post where applicable.

31 Comments »
31 Responses to “Riak and Cassandra and HBase, oh my!”

1.
on 18 May 2010 at 6:34 am elliottcable

I can’t help but wonder why Redis isn’t mentioned. Those are all great tools, but Redis has always been my go-to; Did you simply overlook it, or was it inappropriate for the task in some way?

2.
on 18 May 2010 at 6:54 am deinspanjer

Please keep in mind that as I said, I’m a bit of a technology nut, and I’ve researched and kept track of more than a dozen different NoSQL technologies just because I’m fascinated with the field. I listed the particular three technologies in this post because they seemed to be the closest fits for our project needs.

Redis seems to be a fast and interesting key/value system, but it is very focused on that particular feature set. It stores data in memory to keep it rapidly retrievable whereas we want to focus on persisting the data to disk and performing analysis on it later. Redis still has a lot of work planned on their roadmap as far as sharding and high-availability goes. We needed a solution that could easily provide elastic scaling to suit the time-varying shape of our workloads.

3.
on 18 May 2010 at 8:41 am Sami Samhuri

I don’t think redis scales like Cassandra and Riak. You can’t just spin up nodes, add them to the cluster and walk away. You have to deal with sharding and other hacks to scale horizontally. The nice think about Riak (and probably Cassandra) is that every node is the same, there are no special nodes. If a node goes down, you fire up another node and add it to the cluster. It doesn’t matter which node went down.

This might have changed in Redis-land since I last looked, as things move fast.

4.
on 18 May 2010 at 10:13 am Anonymous

Again shameless promotion but Did you looked at project Voldemort ? Would just like to know your thoughts on what features it was missing making it unsuitable for your project.

5.

on 18 May 2010 at 10:18 am deinspanjer

Again, mostly the fact that Voldemort and Tokyo Cabinet are very focused on a narrower slice of features related to key/value CRUD than more general document storage and analysis. When the MongoDB guys show up and ask me why they weren't on the top three, I'll have a *much* more complicated response. :)

I would encourage anyone who is interested to feel free to reply with the answers for their particular technology interest on each of the given discussion points. Even though our particular use case has been decided, if we collect answers for other candidate technologies here, I hope it might help the NoSQL community in general, and possibly some other team who might have a very similar use case to ours.

6.

on 18 May 2010 at 11:08 am Stas

Hi.

Have you considered MongoDB?

We also reviewed the databases mentioned in the article and really liked its hybrid column-store / RDBMS model.

7.

on 18 May 2010 at 12:00 pm Matt

When considering "manpower" the text seems to imply that using Cassandra's Thrift interface requires more development and testing time than writing a client for Riak's REST interface. Is that what you and the team concluded? If so, why?

Riak keeps looking better, the JavaScript map-reduce support is a good start. I'd be concerned that Riak's map-reduce computations will be significantly slower than Hadoop with Cassandra/HBase.

There are also a good number of tools already available for ad-hoc queries on Hadoop such as Pig, Cascalog, and Hive.

Good luck, I hope you'll share more as the project progresses.

8.

on 18 May 2010 at 12:07 pm deinspanjer

Absolutely wanting to make several posts regarding the project progress.

As for the client dev time, the important consideration there is that we have two very distinct client use-cases. The Test Pilot add-on is a write only client. It already has all the code necessary for it to make a POST request to a particular URL with the needed form data format.

On the analysis side, it is a read only client. We have to spend some time crafting a UI friendly enough for the people who will be using it to get their job done. I suspect the only easy out there would be with something like CouchDB's Futon. My plan is to be able to have a simple web app that uses nothing more than jQuery to craft the necessary POSTs to /mapred for Riak.

What are your thoughts regarding fulfilling those two client requirements in Cassandra?

9.

on 18 May 2010 at 12:30 pm Jonathan

HBase has a builtin(ish) REST server: http://wiki.apache.org/hadoop/Hbase/Stargate

10.

on 18 May 2010 at 12:51 pm Randall

@elliottcable: Redis probably isn't mentioned because there's no native clustering. People who cluster Redis do it with client-side hashing and any redundancy/failover/availability/handoff stuff would have to be hand rolled for now.

Thanks for this post. I've spent the last two weeks evaluating Riak vs Cassandra and I've come to a lot of the same conclusions. I'm just gonna run a benchmark or two on Cassandra for completeness.

To me one of the biggest wins of Riak is that it has a per-bucket backend configuration. If you have some data that only needs memory persistence with low N and other data that needs 4x redundancy and persistence to disk you can do it all on the same cluster. Love it.

11.

on 18 May 2010 at 1:17 pm Amandeep Khurana

HBase master is not a SPOF. Hadoop Namenode is…

12.

on 18 May 2010 at 1:19 pm deinspanjer

That is very true. I should have been more explicit there, but until we get 0.21 with durable writes, and a failover NN, the fact that you could technically run multiple HBase masters is almost a moot point.

13.

on 18 May 2010 at 1:21 pm Anonymous Coward

One of the key selling points of Cassandra is that every node is exactly the same, and there's no single point of failure. That's just not true. There are some nodes that store cluster layout, and if they die, the cluster subsequently modified, then come alive again, then there will be nodes participating in the cluster that are unknown. This causes problems and IIRC, the solution is to do a rolling restart of the cluster. Read the Cassandra mailing list for more detail.

Another more insidious problem with Cassandra is that in fact, every node is a cluster point of failure. Any single node dying kill the entire cluster. Here's how: Cassandra has a concept called hinted handoff. If you write data into the cluster, and the node that's primary for that data is down, then the node that got the write will store that write until the dead node comes back up. If the dead node doesn't come back up, then the node that got the write will eventually fill its disk with HH data, and when that happens, the node also goes dead-mode. Now all remaining nodes start storing HH data for the two dead nodes, and the problem gets worse at double the rate. Eventually the entire cluster will fail.

Yet another problem with Cassandra is that adding a node into the cluster will involve anticompaction and recompaction across existing nodes, which is an extremly I/O-intensive process and I believe requires 2x available disk space in your cluster as your data set size. Further it requires every node have 2x the disk space as the amount of data stored on that node.

I won't go into the Java GC problems of Cassandra.

None of these problems is well-understood. The mailing list participants are all quite quick to brush off such failing points as not worth worrying about, and don't seem eager to delve into such "edge-case" problems to find more-elegant solutions. This in spite of the fact that the existence of such problems come from mailing list participants ACTUALLY HAVING THOSE PROBLEMS.

So at least in the case of Cassandra, there are some serious and subtle problems that require consideration. I imagine Riak has some similar problems but I have far more experience with Cassandra, so I can't speak intelligently to the Riak edge cases.

14.

on 18 May 2010 at 1:34 pm Vagif Verdi

To those suggesting random kv-stores for consideration.

Please keep in mind that the main focus of this comparison seem to be distributed cluster and high availablity. It is NOT about key-value stores or "NoSQL".

So please stop suggesting redis, mongodb and other solutions that do not have automatic sharding and replication.

15.

on 18 May 2010 at 1:38 pm deinspanjer

Well, I know I happened to toss the NoSQL keyword in there a time or two so it would be easy for people to jump in with their tool-du-jour.

MongoDB had a lot going for it in this particular use case given their great ability to index inside arbitrary documents and a similar simple JavaScript based MapReduce model. I really wish that their sharding and replication was well past the "alpha" moniker. :/

16.

on 18 May 2010 at 3:30 pm Lev

FYI: Riak vs. Voldemort performance comparison: http://lionet.livejournal.com/53656.html

17.

on 18 May 2010 at 4:42 pm Imolev

Lev, this is an old benchmark (I believe Riak 0.10 is much faster)

18.

on 19 May 2010 at 6:28 am Jonathan Ellis

Responding to some of the comments:

Cassandra supports replication factor and replication strategy per-keyspace (analogous to riak's buckets).

Anonymous Coward is just plain wrong about almost everything he writes. (The only part with a recognizable kernel of truth is the one about anticompaction requiring extra disk space, which is scheduled to be fixed in 0.7.)

I haven't yet seen a benchmark where Riak got more than 1/5 of Cassandra's performance per machine.

/cassandra developer

19.
on 19 May 2010 at 8:41 am deinspanjer

Thanks very much for responding. Could you share some details or links to the correct facts?

If a machine is down during a cluster change, how is it informed of the change when it comes back up? Do all of the nodes share cluster layout information rather than just some as described?

How does the cluster avoid the suggested hinted handoff problem described here? Will the recipient of a request that requires HH defer the HH to another node if it is overloaded? Is the answer to this problem just to never let a machine stay down or to always have a significant amount of free disk space?

20.
on 21 May 2010 at 12:04 am Andres

Go get postgresql, that should have no trouble supporting the load. If you want high availability, set up replication with Londiste (http://wiki.postgresql.org/wiki/Londiste_Tutorial). If you want fast response times, get https://developer.skype.com/SkypeGarage/DbProjects/PlProxy.

21.
on 21 May 2010 at 8:04 am deinspanjer

@Andres
Do you have experience with PLProxy? Do you know if it can shard out insert statements? Do you have a guess as to how many master shards would be needed to support the peak write loads? Since PostgreSQL isn't elastic, we'd have to commit to always having enough machines available for that peak load in traffic. Also, if we ever needed to grow the cluster, we'd have to write a system that could rehash and rebalance all the data. We'd need to be able to store the several TB of data, what would you recommend there, SAN? We'd also need the HA you mentioned so each of those masters would need hot fail-over slaves. Any idea if PLProxy supports that type of configuration?
Was this suggestion based on a serious review of the project requirements or just tossing some suggestions out there or maybe just lashing out at non-RDBMS solutions?

22.
on 24 May 2010 at 11:34 am Nick Kallen

I work for a company that had to turn off hinted handoff for reasons similar to but not identical to what Anonymous Coward described. We've also experienced serious issues with GC.

Cassandra is really awesome and promising. At the same time it has often been unstable in our production environment. I don't think this makes Cassandra less of a contender than Riak. It just means this whole category of tools ought to be approached as beta software. This is a great opportunity for a risk-tolerant programmer to be part of an exciting emerging technology.

There was a great post by Reddit recently that described their experience with RabbitMQ, which was the hawt technology of the minute this time last year. Suffice to say it has issues. If anything, I trust Riak less than Cassandra–but I'm just superstitious. We've had mostly bad experiences with the Erlang runtime (ejabberd, rabbitmq).

That said, you shouldn't blame the tools. But the strange irony software at scale is: what works works. You can say all you want about HBase having a single point of failure but it's totally beside the

point. If you can't run a p2p system with high availability because the code has bugs then what are we talking about?

I sound like a Cassandra/Riak hater and I'm not. I think these are huge, foundational technologies of the future Internet. Just want to share my experience.

23.
on 09 Jun 2010 at 1:49 pm Henry Ho

why not CouchDB btw? Sorry, just a nosql newbie.

24.
on 09 Jun 2010 at 1:52 pm deinspanjer

Shortest answer is because CouchDB isn't elastic in the same way as the engines discussed above. While there is some interesting work being done in Mozilla with CouchDB for random little side projects, we've had some unfortunate experiences with trying to use it for larger datasets and in production environments.

25.
on 16 Jul 2010 at 12:52 pm Cassandra, HBase, Riak: Choosing the Right Solution. › PHP App Engine

[...] Cassandra, HBase, Riak: Choosing the Right Solution: Mozilla shows us the right way of choosing a storage solution (as opposed to this completely [...]

26.
on 17 Jul 2010 at 11:09 am mikeal

I work on/with CouchDB all day every day.

CouchDB would *not* work for this use case and isn't trying to optimize for it. This use case is exactly what Cassandra, Riak, and HBase are going for and this writeup goes a long way towards evaluating their ability to handle it.

Great post guys :)

Also, many of those bad experiences with CouchDB at Mozilla were related to the CentOS VM not having the proper VMWare tools installed and decided to just kill the CouchDB erlang process all the time. CouchDB is being used for some crash reproduction automation and automation results systems and it runs fine once we figured out the vmware issue.

27.
on 23 Jul 2010 at 4:05 pm Eamonn O'Brien-Strain » links for 2010-07-23

[...] Blog of Data » Blog Archive » Riak and Cassandra and HBase, oh my! Interesting article and comments looking at scalable key-store data stores like Riak, HBase, and Cassandra. (tags: architecture database scaling) [...]

28.
on 31 Jul 2010 at 9:00 pm Salman

Nothing like another nosql debate, like always run your own tests and don't believe the hype :)

29.
on 02 Aug 2010 at 4:35 am Diego Caravana

Thanks for this interesting and useful article (and also for the comments). I'm researching tools for a similar project, and I love MongoDB so much with automatic sharding and availability almost ready in the not-yet-released 1.6 version. I've tried Cassandra (a good experience indeed) and read something about HBase, but now I think Riak deserves some attention, and Hypertable, too.

30.

on 13 Aug 2010 at 10:39 am Alternative Database Technology for the Cloud: There is No Silver Bullet | Rackspace Cloud Computing & Hosting

[...] of a few database alternatives, but many more also exist. It also links to the write-up about Mozilla's Test Pilot project, where they talk about the process they used to select a database that met their [...]

31.

on 16 Aug 2010 at 9:34 pm Blog of Data » Blog Archive » Benchmarking Riak for the Mozilla Test Pilot Project

[...] the experiment results and performing analysis on them. As discussed in the previous blog post, Riak and Cassandra and Hbase, oh my!, we decided on Riak as that [...]