Filling in the BigTable Gaps

Big Data is big innovation, big headaches and in the end, big money. The only problem is, is it can be a huge pain to get running….and then to get running correctly. Recently – in the last 5-8 years – we have seen huge efforts by major companies (Yahoo, Facebook, Twitter, etc.) have put a lot of resources behind these technologies, doubling down on certain stacks to enable next generation business. So there is definitely something there.

The technology is still very immature and driven almost entirely by the Open source community. The implications of that is another blog post [1], but the punch line is there are a lot of rough edges, incomplete features and frequently a sloooow process [2].

More than a few companies have also been started (Cloudera, Datastax, Opscode, Hortonworks, etc.) around the ideas of to make these tools stable, fast and enterprise ready. Oh, and they sell support (gotta make money somehow, right?). So clearly this cloud stuff needs a lot of help and a lot of more features.

But I’m getting ahead of myself.

Lets jump back 5 years – Google releases the ([BigTable paper] (labs.**google**.com/papers/**bigtable**-osdi06.pdf)) and the open source world jumped at the idea, quickly spinning up ([HBase] (org.apache.hbase)) under the Hadoop umbrella. And for a while it seemed great! I can store petabytes of data – awesome. I can access it in real time – even better. And do appends, updates and deletes over a write-once file system? Fantastic. It was even so great the US Government put their heads together and came up with their own version of BigTable, ([Accumulo] (<http://incubator.apache.org/accumulo/>)), optimized for high throughput, though still faithful to many of the aspects of the original BigTable.

So great, we have this massively scalable database. Well, turns out that BigTable doesn’t cover everything we want to do with the database, particularly if you want to do fast lookups or scale out even farther or do traditional RDMS operations. So along comes Megastore (research.google.com/pubs/archive/36971.pdf). Now, there are a lot of things going on in Megastore that most of the companies outside of Google don’t need or are covered via alternative means (see ([Hive] (hive.apache.org)) or ([Pig] (pig.apache.org))). However, one of the things that isn’t really covered by external tools is indexing.

Now you are probably, “Woah, hold on! What about ([Lily] (<http://www.lilyproject.org/lily/index.html>))? Or ([Solr] (<http://lucene.apache.org/solr/>))? Or etc….???” Well these things are good if you are doing indexing on just one thing – unstructured text in a given field. And a lot of times, that’s all you need. This is especially true as these tools integrate with search tools. However, what about the case where you need to index across multiple fields? Or build your own special indexes to make it go fast? What about trying out new index schemes? Then you are going to be out of luck and hand rolling your own.

To make sure these indexes scale you then have to store them in a cloud (probably the same database as the one hosting your data). Okay, doable but that can get a little tricky to make sure it scales well. Then you have to make sure that when you update your database that the indexes also get updated. And then you have to build a tool to use those indexes. What about using something SQL-like? Then you are writing your own SQL parser and then pipe that into your indexing and *then* use that to pull out and combine data. Now consider that you have to make that performant on a cloud scale. Ouch.

Clearly this is a hard problem. And every organization doesn’t want to solve this problem from the ground up, scalably every time.

You just want to write indexes and have it integrate with all your tools right away. You don’t want to deal with writing a specific client to handle indexing your data on ingest. You don’t want to have to worry about using those indexes on query.

You just want to get some data out as fast a possible. And with standard BigTable, this isn’t possible. Yeah, it’s pretty fast. And yeah, it scales like crazy. But you need to do a lot of work to make sure it goes fast. And you *need* it to go fast.

Enter Culvert.

Culvert is a secondary indexing *platform*, which means it provides everything you need to write indexes and use them to access and well, index your data.

It takes care of all the pains of indexing your data as it comes in –you don’t need to worry about making sure your index tables match your real table. Culvert ensures that when you do a query, indexes are used properly to get you back the answer as fast as possible.

All you have to do is write your own indexes so your data can be accessed quickly. Cut out all the developers to build a custom interaction. Drop all the people worrying about maintaining a special database for the indexes. All you need is a couple smart people with a good idea of what the data looks like to write down the best way to access the data.

Sounds easy, right?

In the ([next post] (jyates.github.com/ 2011/11/11/welcome-to-index-nirvana.html)) I’ll talk about how you can actually use Culvert in your own system

[1] Working on that post.

[2] Not always – there are many cases where the open source stuff is way better than the closed version. However, this tends to be the exception, not the rule. It is interesting that in the cloud space, open source software has proven to be far more widespread (and higher quality) than the closed source solutions. For the counterexample, see ([MapR] (http://mapr.com/))