

Lars George Meeting notes

- Lars
 - Worked 5 years in Australia, 4 1/2 in Vegas, now in Germany
 - Started in Delphi, Pascal, ...
 - Uses HBase since 2007
- Built and maintains Worldlingo using HBase
 - Machine & Human translation
 - Used by Windows and European Patent Office (EPO)
- Built translation application for Patents.com
 - 420 mio documents
 - Uses HBase, Lucene ; now looking to use Katta for sharding the search index
 - Uses 44 physical machines, 88 virtual Xen machines
 - 1 'commodity' hardware machine = 12GB RAM, 2x1TB disk
 - 3.5GB used to run Tomcat, 8.5GB to run HBase, Hadoop
 - Lucene index is currently 190GB (only a fraction (20 mio) of the documents is currently in the system)
 - on a local drive
 - uses about 6GB of heap
 - tried to put it in Hadoop HDFS but that's too slow
 - He's combining different fields to put in the Lucene index
 - When a new combination of fields would be chosen, a new M/R job needs to be run to update the index.
 - The index combines about 45 fields out of the docs
 - A full rebuild of the index with M/R takes about 24hours
 - Thinking about Kata to use for a sharded index
 - It uses standard Lucene indexes underneath
 - Which is not the the case with Solr, and it's not clear where Solr is heading
- Avoid problems with small clusters
 - Minimum for a HBase cluster is 10 nodes
 - A lot of problems with a cluster of only 6 nodes
 - There's one master node which does almost nothing
 - 5 data nodes (region servers) have to handle all the work.
 - Hbase region server
 - Hadoop hdfs
 - M/R Tasktracker
 - 2 M/R Map jobs
 - 2 M/R Reduce jobs
 - Zookeeper
 - If one region server can't keep up with the work it fails, leaving only 4 nodes to take over its regions and subsequently handle more work. Those nodes will in turn also start failing. And finally a domino effect will bring down the whole system.
 - A failing region server is not so big an issue on a large cluster since there are enough other servers to take over the work.
 - E.g. if you have 1000 regions on one regionserver and it fails, 1000 regions need to be distributed over the remaining servers and the write-ahead log to be split into 1000 pieces.

More servers with each only 100 regions will result in only 100 regions to be redistributed when one regionserver fails.

- Problems occur especially when stressing the system with bulk imports, not so much under 'normal' web-read use.
 - Alternative approach : import data directly into HFiles
 - First calculate the region splits
 - Then import into HFiles
 - Put the HFiles and corresponding regions in the .META. regionfiles
 - Then start hbase which picks up the HFiles
 - Do a Major compaction once a day
- Hbase <0.19 was not stable, since 0.20 really stable
 - Hadoop hdfs is really stable. The problems were especially in the combination of HBase and Hadoop, not in Hadoop itself.
- Problems during upgrade were mainly due to 'human' mistakes rather than HBase problems
 - Having a backup system for an iterative upgrade approach is a good idea, but a lot of hardware is needed !
 - First do a hdfs copy of the data to the 2nd system
 - Then upgrade the 2nd system
 - If all went well, switch to this system
- A region split is fast
 - It is only putting pointers that point to the start of the region in the HFiles
- A Major compaction is more work
 - It merges HFiles into one file
 - In case of a region split : it does the actual split into two new files
 - Meanwhile new ingest only goes to the memcache which waits to flush until the compaction is finished
 - If it takes a long time and the memcache reaches its limit, the region server dies
- # open files should be put from 1K to 32K on the OS
 - especially on a production system
 - the start of the hbase logfile shows the value in use
- If possible put each system on its own server
 - Each subsystem has a lot of logs going on
 - Zookeeper does not need 'heavy' machines, but needs a quorum to stay available
 - If Zookeeper is running on a node that also has hbase, hdfs, M/R it can become unresponsive and bring down the system.
 - Better to use a separate (cheap) set of machines.
 - At least put it on a different disk or partition than hbase/hdfs
 - Map/Reduce on HBase can optimize to run map jobs on same server as region, but that's not where the actual data is stored (hdfs). So not much to be gained there.
- RabbitMQ is used to add info to the Lucene index
 - One RabbitMQ server seems enough. No sharding is needed.
 - The messages in the queue are really small (few bytes) and allow for fast adding to the queue.
 - The message is a small string that describes what needs to be done (e.g. which data to combine and put in the index)
 - The RabbitMQ setup can promise that no messages will be skipped, but some messages might be picked up twice.

- Make sure to one of the more recent RabbitMQ versions with QOS
- For Lucene index, a thread wakes up once a day and processes the queue
 - This way only one or two extra index files a day are created and not too many compactions a day are needed
- 2-ary indexes are created by the application
 - If transactional behaviour is important, the application can lock the rows itself
 - The index/transactional package of HBase could also be used, but even this doesn't promise 100% transactional behaviour.
- There's no sorting of columns like in Cassandra
 - You can request all columns of a column family and then sort them yourself
- Our architecture looks very similar as the one Lars is using. He has no major remarks on it.
- Our mapping of the document model onto the hbase schema looks fine too.
 - But we have to be carefull when using the timestamps, although our way of using them could indeed work.
 - HBase can get confused when writing cells with ' older ' timestamps
 - When updating a cell with a new value, but with the same (existing) timestamp, it is undefined which value will actually be returned and kept after a compaction of the HFiles.
 - Make sure that there's only one timestamp generator or/and that the clocks of the clients/ servers are synchronized etc.
- Having columns that are almost never updated next to columns that change often (f.i. with each document version) is not a big issue
 - All cells of a row are next to each other inside an HFile.
 - 'Old' values can be in an older HFile, but those HFiles are merged/compacted once a day. So it's not that you would be scanning tons of HFiles to find an 'old' value.
- Each table in HBase has its own regions.
- Be aware that when defining a range, f.i. to perform a row-scan, the last item of the range is exclusive. So, a range [A,C[would return A and B but not C.
- There's a lack of good comprehensive logging and log inspection tools.
 - If a cluster of 40nodes crashes you have numerous of GBs of logs to dive into to find out what happened.
- **" You have to build for failure "**