Item 20
git_comments:

1. !/bin/sh
2. word_count Jar.
3. Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
4. Cassandra class files.
5. !/bin/sh
6. word_count Jar.
7. Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
8. Cassandra class files.
9. ~ Licensed to the Apache Software Foundation (ASF) under one ~ or more contributor license agreements. See the NOTICE file ~ distributed with this work for additional information ~ regarding copyright ownership. The ASF licenses this file ~ to you under the Apache License, Version 2.0 (the ~ "License"); you may not use this file except in compliance ~ with the License. You may obtain a copy of the License at ~ ~ http://www.apache.org/licenses/LICENSE-2.0 ~ ~ Unless required by applicable law or agreed to in writing, ~ software distributed under the License is distributed on an ~ "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY ~ KIND, either express or implied. See the License for the ~ specific language governing permissions and limitations ~ under the License.
10. this is enough for testing a single server node; may need more for a real cluster
11. Let ToolRunner handle generic command-line options
12. * * This counts the occurrences of words in ColumnFamily Standard1, that has a single column (that we care about) * "text" containing a sequence of words. * * For each word, we output the total number of occurrences across all texts.
13. text0: no rows
14. text3: 1000 rows, 1 word
15. text2: 1 row, 2 words
16. text1: 1 row, 1 word
17. ~ Whether or not to use a framed transport for Thrift. If this option ~ is set to true then you must also use a framed transport on the ~ client-side, (framed and non-framed transports are not compatible).
18. ~ Buffer size to use when flushing memtables to disk. (Only one ~ memtable is ever flushed at a time.) Increase (decrease) the index ~ buffer size relative to the data buffer if you have few (many) ~ columns per key. Bigger is only better _if_ your memtables get large ~ enough to use the space. (Check in your data directory after your ~ app has been running long enough.)
19. ~ Time to wait before garbage-collection deletion markers. Set this to ~ a large enough value that you are confident that the deletion marker ~ will be propagated to all replicas by the time this many seconds has ~ elapsed, even in the face of hardware failures. The default value is ~ ten days.
20. ~ Flush memtable after this much data has been inserted, including ~ overwritten data. There is one memtable per column family, and ~ this threshold is based solely on the amount of data stored, not ~ actual heap memory usage (there is some overhead in indexing the ~ columns).
21. ~ ColumnFamily definitions have one required attribute (Name) ~ and several optional ones. ~ ~ The CompareWith attribute tells Cassandra how to sort the columns ~ for slicing operations. The default is BytesType, which is a ~ straightforward lexical comparison of the bytes in each column. ~ Other options are AsciiType, UTF8Type, LexicalUUIDType, TimeUUIDType, ~ and LongType. You can also specify

the fully-qualified class ~ name to a class of your choice extending ~ org.apache.cassandra.db.marshal.AbstractType. ~ ~ SuperColumns have a similar CompareSubcolumnsWith attribute. ~ ~ BytesType: Simple sort by byte value. No validation is performed. ~ AsciiType: Like BytesType, but validates that the input can be ~ parsed as US-ASCII. ~ UTF8Type: A string encoded as UTF8 ~ LongType: A 64bit long ~ LexicalUUIDType: A 128bit UUID, compared lexically (by byte value) ~ TimeUUIDType: a 128bit version 1 UUID, compared by timestamp ~ ~ (To get the closest approximation to 0.3-style supercolumns, you ~ would use CompareWith=UTF8Type CompareSubcolumnsWith=LongType.) ~ ~ An optional `Comment` attribute may be used to attach additional ~ human-readable information about the column family to its definition. ~ ~ The optional KeysCachedFraction attribute specifies ~ The fraction of keys per sstable whose locations we keep in ~ memory in "mostly LRU" order. (JUST the key locations, NOT any ~ column values.) The amount of memory used by the default setting of ~ 0.01 is comparable to the amount used by the internal per-sstable key ~ index. Consider increasing this if you have fewer, wider rows. ~ Set to 0 to disable entirely. ~ ~ The optional RowsCached attribute specifies the number of rows ~ whose entire contents we cache in memory, either as a fixed number ~ of rows or as a percent of rows in the ColumnFamily. ~ Do not use this on ColumnFamilies with large rows, or ~ ColumnFamilies with high write:read ratios. As with key caching, ~ valid values are from 0 to 1. The default 0 disables it entirely.
22. ~ Throughput setting for Binary Memtables. Typically these are ~ used for bulk load so you want them to be larger.
23. ~ The address to bind the Thrift RPC service to. Unlike ListenAddress ~ above, you *can* specify 0.0.0.0 here if you want Thrift to listen on ~ all interfaces. ~ ~ Leaving this blank has the same effect it does for ListenAddress, ~ (i.e. it will be based on the configured hostname of the node).
24. === Basic Configuration ===
25. ~ Interval at which to perform syncs of the CommitLog in periodic mode. ~ Usually the default of 10000ms is fine; increase it if your i/o ~ load is such that syncs are taking excessively long times.
26. === Memory, Disk, and Performance ===
27. ~ CommitLogSync may be either "periodic" or "batch." When in batch ~ mode, Cassandra won't ack writes until the commit log has been ~ fsynced to disk. It will wait up to CommitLogSyncBatchWindowInMS ~ milliseconds for other writes, before performing the sync. ~ This is less necessary in Cassandra than in traditional databases ~ since replication reduces the odds of losing data from a failure ~ after writing the log entry but before it actually reaches the disk. ~ So the other option is "timed," where writes may be acked immediately ~ and the CommitLog is simply synced every CommitLogSyncPeriodInMS ~ milliseconds.
28. ~ Access mode. mmapped i/o is substantially faster, but only practical on ~ a 64bit machine (which notably does not include EC2 "small" instances) ~ or relatively small datasets. "auto", the safe choice, will enable ~ mmapping on a 64bit JVM. Other values are "mmap", "mmap_index_only" ~ (which may allow you to get part of the benefits of mmap on a 32bit ~ machine by mmapping only index files) and "standard". ~ (The buffer size settings that follow only apply to standard, ~ non-mmapped i/o.)
29. ~ The maximum number of columns in millions to store in memory per ~ ColumnFamily before flushing to disk. This is also a per-memtable ~ setting. Use with MemtableThroughputInMB to tune memory usage.
30. ~ If you are using an order-preserving partitioner and you know your key ~ distribution, you can specify the token for this node to use. (Keys ~ are sent to the node with the "closest" token, so distributing your ~ tokens equally along the key distribution space will spread keys ~ evenly across your cluster.) This setting is only checked the first ~ time a node is started. ~ This can also be useful with RandomPartitioner to force equal spacing ~ of tokens around the hash space, especially for clusters with a small ~ number of nodes.
31. Local hosts and ports
32. ~ Address to bind to and tell other nodes to connect to. You _must_ ~ change this if you want multiple nodes to be able to communicate! ~ ~ Leaving it blank leaves it up to InetAddress.getLocalHost(). This ~ will always do the Right Thing *if* the node is properly configured ~ (hostname, name resolution, etc), and the Right Thing is to use the ~ address associated with the hostname (it might not be).
33. ~ Addresses of hosts that are deemed contact points. Cassandra nodes ~ use this list of hosts to find each other and learn the topology of ~ the ring. You must change this if you are running multiple nodes!

34. ~ Add column indexes to a row after its contents reach this size. ~ Increase if your column values are large, or if you have a very large ~ number of columns. The competing causes are, Cassandra has to ~ deserialize this much of the row to read a single column, so you want ~ it to be small - at least if you do many partial-row reads - but all ~ the index data is read for each access, so you don't want to generate ~ that wastefully either.
35. internal communications port
36. <CommitLogSyncBatchWindowInMS>1</CommitLogSyncBatchWindowInMS>
37. ~ Directories: Specify where Cassandra should store different data on ~ disk. Keep the data disks and the CommitLog disks separate for best ~ performance
38. Thrift RPC port (the port clients connect to).
39. ~ The name of this cluster. This is mainly used to prevent machines in ~ one logical cluster from joining another.
40. ~ Unlike most systems, in Cassandra writes are faster than reads, so ~ you can afford more of those in parallel. A good rule of thumb is 2 ~ concurrent reads per processor core. Increase ConcurrentWrites to ~ the number of clients writing at once if you enable CommitLogSync + ~ CommitLogSyncDelay.
41. Size to allow commitlog to grow to before creating a new segment
42. ~ Delay (in milliseconds) during which additional commit log entries ~ may be written before fsync in batch mode. This will increase ~ latency slightly, but can vastly improve throughput where there are ~ many writers. Set to zero to disable (each entry will be synced ~ individually). Reasonable values range from a minimal 0.1 to 10 or ~ even more if throughput matters more than latency.
43. Time to wait for a reply from other nodes before failing the command
44. Number of replicas of the data
45. ~ Partitioner: any IPartitioner may be used, including your own as long ~ as it is on the classpath. Out of the box, Cassandra provides ~ org.apache.cassandra.dht.RandomPartitioner, ~ org.apache.cassandra.dht.OrderPreservingPartitioner, and ~ org.apache.cassandra.dht.CollatingOrderPreservingPartitioner. ~ (CollatingOPP colates according to EN,US rules, not naive byte ~ ordering. Use this as an example if you need locale-aware collation.) ~ Range queries require using an order-preserving partitioner. ~ ~ Achtung! Changing this parameter requires wiping your data ~ directories, since the partitioner can modify the sstable on-disk ~ format.
46. Miscellaneous
47. ~ Licensed to the Apache Software Foundation (ASF) under one ~ or more contributor license agreements. See the NOTICE file ~ distributed with this work for additional information ~ regarding copyright ownership. The ASF licenses this file ~ to you under the Apache License, Version 2.0 (the ~ "License"); you may not use this file except in compliance ~ with the License. You may obtain a copy of the License at ~ ~ http://www.apache.org/licenses/LICENSE-2.0 ~ ~ Unless required by applicable law or agreed to in writing, ~ software distributed under the License is distributed on an ~ "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY ~ KIND, either express or implied. See the License for the ~ specific language governing permissions and limitations ~ under the License.
48. ~ Strategy: Setting this to the class that implements ~ IReplicaPlacementStrategy will change the way the node picker works. ~ Out of the box, Cassandra provides ~ org.apache.cassandra.locator.RackUnawareStrategy and ~ org.apache.cassandra.locator.RackAwareStrategy (place one replica in ~ a different datacenter, and the others on different racks in the same ~ one.)
49. ~ Authenticator: any IAuthenticator may be used, including your own as long ~ as it is on the classpath. Out of the box, Cassandra provides ~ org.apache.cassandra.auth.AllowAllAuthenticator and, ~ org.apache.cassandra.auth.SimpleAuthenticator ~ (SimpleAuthenticator uses access.properties and passwd.properties by ~ default). ~ ~ If you don't specify an authenticator, AllowAllAuthenticator is used.
50. ~ Turn on to make new [non-seed] nodes automatically migrate the right data ~ to themselves. (If no InitialToken is specified, they will pick one ~ such that they will get half the range of the most-loaded node.) ~ If a node starts up without bootstrapping, it will mark itself bootstrapped ~ so that you can't subsequently accidently bootstrap a node with ~ data on it. (You can reset this by wiping your data and commitlog ~ directories.) ~ ~ Off by default so that new clusters and upgraders from 0.4 don't ~ bootstrap immediately. You should turn this on when you start adding ~ new nodes to a cluster that already has data on it. (If you are upgrading ~ from 0.4, start your cluster with it off once before changing it to true. ~ Otherwise, no data will be lost but you will incur a lot of unnecessary ~ I/O before your cluster starts up.)
51. ~ Keyspaces and ColumnFamilies: ~ A ColumnFamily is the Cassandra concept closest to a relational ~ table. Keyspaces are separate groups of ColumnFamilies. Except in ~ very unusual circumstances you will have one Keyspace per application. ~ There is an implicit keyspace named 'system' for Cassandra internals.

52. ~ EndPointSnitch: Setting this to the class that implements ~ AbstractEndpointSnitch, which lets Cassandra know enough ~ about your network topology to route requests efficiently. ~ Out of the box, Cassandra provides org.apache.cassandra.locator.EndPointSnitch, ~ and PropertyFileEndPointSnitch is available in contrib/.
53. ~ Buffer size to use when performing contiguous column slices. Increase ~ this to the size of the column slices you typically perform. ~ (Name-based queries are performed with a buffer size of ~ ColumnIndexSizeInKB.)
54. ~ The maximum time to leave a dirty memtable unflushed. ~ (While any affected columnfamilies have unflushed data from a ~ commit log segment, that segment cannot be deleted.) ~ This needs to be large enough that it won't cause a flush storm ~ of all your memtables flushing at once because none has hit ~ the size or count thresholds yet. For production, a larger ~ value such as 1440 is recommended.
55. for hadoop
56. * * Try the storage-config system property, and then inspect the classpath.
57. try the classpath

git_commits:

1. **summary:** add wordcount hadoop example
   message: add wordcount hadoop example patch by jbellis; reviewed by Stu Hood for CASSANDRA-342 git-svn-id: https://svn.apache.org/repos/asf/incubator/cassandra/trunk@909626 13f79535-47bb-0310-9956-ffa450edef68

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** hadoop integration
   description: Some discussion on -dev: http://mail-archives.apache.org/mod_mbox/incubator-cassandra-dev/200907.mbox/%3Cf5f3a6290907240123y22f065edp1649f7c5c1add491@mail.gmail.com%3E

jira_issues_comments:

1. This patch adds the ability for Cassandra databases to be read from in a Hadoop setting. This is the "stupid" version of said support (c.f. "not-stupid" discussion http://mail-archives.apache.org/mod_mbox/incubator-cassandra-dev/200907.mbox/%3Cf5f3a6290907240123y22f065edp1649f7c5c1add491@mail.gmail.com%3E). This patch is only working when run in the non-distributed `./bin/hadoop jar` environment of Hadoop. .h2 Building Building the patched Cassandra requires including `hadoop-0.20.0-core.jar` that is distributed with Hadoop 0.20.0 (obviously) in Cassandra's `$CLASSPATH`. (Which is easiest to do by simply copying the file to Cassandra's `lib` directory.) An example of the adapter's use can be found in `src/examples/org/apache/cassandra/examples/WordCount.java`. You can run `WordCount.java` with Hadoop by first editing the `conf/hadoop-env.sh` file and adding all of the jars in Cassandra's lib directory to `$HADOOP_CLASSPATH`. Building the examples is then just a matter of running `ant examples`. .h2 Running Running the example is straightforward after that. Assuming you've added some tweets to the Tweets column family with a column called `text` filled with, you know, text: {code} ./bin/hadoop jar \ /path/to/cassandra/build/apache-cassandra-incubating-examples-0.4.0-dev.jar\ org.apache.cassandra.examples.WordCount -inputspace Twitter \ -inputfamily Tweets -outputdir outie \ -confdir /path/to/cassandra/src/examples/conf/wordcount/ {/code} .h2 Changes External to `cassandra.hadoop` This patch makes two changes in the Cassandra project that are outside of the new `hadoop` package. # Makes `StorageProxy.getKeyRange()` public. # `RowSerializer` is now a public class and outside of Row. This was done so I didn't have to rewrite the serialization code for writing the `RowWritable` class. # Adds the `examples` ant task for building the jar file of cassandra example code.

.h2 Issues This patch does have some issues. Specifically: # Has no tests. # Cannot split up the key ranges beyond what the entire key range that exists on each individual node. This means we cannot delegate to more Map tasks than there are Cassandra nodes. As we move to billions of keys per node, this is even more of an issue. (c.f. CASSANDRA-242) # Cassandra currently must be booted by this Hadoop-facing code in order to work as a side effect of needing certain internal calls in odd places and the onus put upon this project to keep everything working internally. There is currently no way to hook into an external Cassandra process. # Only has been tested and only works (due to the above boot code issues) on one Cassandra node, with one Hadoop Map task. # Cannot take key ranges that cross over multiple nodes. This is a problem with how we (can't) divvy up the keys instead of any other problem (such as the ones described in CASSANDRA-348). # The current API for selecting what keys to grab cannot take anything more than the table/keyspace to search in and the name of a top-level super column. # Because of the lack of true "multiget" support, the reads from the database have a round trip cost for each key desired. # The API is not well-fleshed out for grabbing data from a RowWritable. # Only provides read capability, no writing. # `KeyspaceRecordReader#getProgress()` is nothing more than a stub. # `RowWritable` does not implement `WritableComparable`, which would allow its use as a key and not just a value in a MapReduce job. # `RowWritable` uses `RowSerializer` which encodes way too much information about the column families and columns through `ColumnFamilySerializer`. # Has a (likely inescapable) dependency on the hadoop 0.20 core jar. # Really, really has no tests. I could go into more detail about some of these issues, but this is already too long and the discussion adds even more text.
2. I could also provide an unsquashed version of the commits that led to this patch, if needed. I didn't do it because the task seem a bit arduous for the 14 commits involved and git-jira-attacher seems to not work at the moment.
3. as a non-hadoop expert it would help a lot with review to unsquash. what errors are you getting with g-j-a? Us python guys can probably help with that :)
4. Corrected version of earlier squashed patch. Unsquashed will happen as soon as git-jira-attacher.py works for me.
5. The unsquashed version of the hadoop commits.
6. awesome! now could you squash a little? :) bear with me -- what we want to see is separate functionality in different patches (e.g. 02, 10) but not the evolution within those (04 should be squashed onto 01, i think). (what I do with large patchsets is, i keep the original "raw" patches, then branch that for rebasing stuff around. sometimes I end up with 7 or 8 branches before it's done. :)
7. 0001) Provides a few changes needed to support the new Hadoop bridge. Specifically, StorageProxy.getKeyRange() becomes public, RowSerializer becomes accessible to the hadoop package classes, and Token#originalToken() is added. 0002) This adds the actual Hadoop bridge code. It includes subclasses of InputFormat, RecordReader, and InputSplit. It also provides a nice RowWritable class to be used as the value passed to a map. This is also the commit that adds the really nasty boot up code in the class BootUp. 0003) This just adds a simple WordCount example of the Cassandra/Hadoop bridge.
8. So, my biggest problem with this patch right now is the boot up code and the way it combines with the local-only query code. It forces us into booting a brand new cassandra instance that assumes the data is already there and ready for the taking but only when a MapReduce task is being done. This is all sorts of bad news. There does not seem to be a way of getting to the internals of Cassandra we need (reading from and writing to the disk and memtable, figuring out what keys are on what nodes, etc.) without also having to boot all of the various Cassandra services. I'm looking for input on how we can get around that. FYI, the HBase way is to have HBase running on the machine already and throw up a connection to it from another process that is created with the information from the InputSplit (on the map task machines) and from the config files (on the initial machine that creates the InputSplits).
9. DataOutputStream implements DataOutput -- can you just switch RowSerializer to ICompactSerializer2, which only requires the latter? (goal is eventually get rid of ICompactSerializer, then rename 2)
10. we can clean the bootup stuff a lot. let's take the bootupunsafe method that shares code w/ cassandra daemon, make it a public static method there, move the LogUtil.init call into StorageService.init, and remove CassandraServer.init. that will be useful for other people looking to embed cassandra too. style note: inline throw-away variables like String ll. + keyspace = DatabaseDescriptor.getTable(keyspace); + if (keyspace == null) inline that too rather than re-binding keyspace to a different meaning (although of the same type) there are a lot of FIXME but it looks like a good start :)
11. i'm a little confused by the bootup business though -- if this code is running on a cassandra node, can't it just use the already-running server?
12. only comment on the example is you should probably strip out the comments from the sample xml copy.
13. btw, i think this was a good patch breakdown.

14. Okay, so here's a new set of patches that we'll call v3. ICompactSerializer2 is now used by RowSerializer (I had forgotten it existed!). The throw-away variables have been tossed. The WordCount storage-conf.xml has been edited down. And the call to CalloutManager has been removed as it no longer exists in trunk. The boot code I'm going to discuss in my next comment.
15. Okay, before we talk about the boot code, let me address some of the confusion about Hadoop. In Hadoop, there are things called Jobs, which are a combination of a Map and a Reduce operation and the InputFormat configuration you specify which are then run across a bunch of machines. A Task is an individual Map or Reduce operation run on one of those machines (so every Job has many Tasks). For every new Task needed, a new JVM is booted up.[1] This is actually okay, distributed-systems-wise, because it keeps all the Tasks from interfering with one another. It does, however, make our jobs harder. There is no way for a Task (and thus this Hadoop code in these patches) to access the runtime of a Cassandra node already on the machine because they will be in separate JVMs! HBase, as I mentioned above, solves this problem by first starting up HBase on those remote machines, and then having each Task create an HTable object from the InputSplit handed to it. This HTable object connects to the local HBase process. (Of course, this same thing happens in the JVM that creates the InputSplits.) So, here's my deal. There is no way for this currently designed system to work efficiently in a distributed system. This is because we have to boot a brand new Cassandra process on machines that might already have (and need if hardware is limited) one running already. The boot up time for Cassandra alone is a big time sink. And consider how these nodes would interoperate with the "stable", non-Hadoop nodes that would start sending them data. Ugh. We can avoid all of this boot time drama if we can come up with a good way of remotely accessing all of the internal information we need from the Cassandra node already running. I have not been able to come up with an alternative solution. Comments? [1] There is something called "Task reuse" that can be configured into a Hadoop deployment. However, the "reuse" only means that a Task can be used more than once for the same Job. So, it's basically just another complicating factor in our boot loading code (one of the reasons there is BootUp.boot() and BootUp.bootUnsafe()) but doesn't help us with our problem.
16. thanks, all of that makes sense now. I'm +1 on applying what Jeff has here as a first step and refining in another ticket. Anyone else have feedback?
17. Since the Hadoop InputFormat can't possibly be in the same process as the Cassandra server on the local machine, perhaps the cleanest interface between them would be a 'private' client library (not using Thrift) that allows for the calls you need? The only call I see to an internal API is StorageService.getRangeToEndPointMap, so we could add a Verb that queries a remote/local running node for the same information. It doesn't look like it would be too difficult to extract just the net.MessagingService portion of the Cassandra server, and then use that as the private client interface to send messages to other nodes.
18. We already have a proposal from Jun over in #197 to expose the ring as a string property, and (presumably) load it into a storageservice. Which would more or less take care of things, although the separation could be cleaner.
19. sorry, CASSANDRA-197
20. Good call... the RingCache mentioned on CASSANDRA-197 is exactly what this ticket needs.
21. Yep, that is exactly what I was thinking about. I'll nose myself around over there.
22. I take back what I said about RingCache: you'll still need to be able to send a RangeCommand, which requires a MessagingService. I suggest we take the approach of embedding a MessagingService like CASSANDRA-337 does.
23. Why can't the recordReader get the rows in a range through thrift? This can be done by first calling get_key_range, followed by a bunch of gets, one for each row.
24. So, should we commit this as a useful first step? (Edit: I think the answer is yes, but the question has become academic as the latest patches no longer apply.)
25. For posterity, a link to Jeff's longer explanation of where he was going here: http://markmail.org/message/5qou35zzdv7uzup6
26. To get around the hadoop-stuff-has-to-run-in-a-different-JVM problem: what if we had Hadoop operate on Cassandra snapshots? For the kind of batch oriented, non-latency-sensitive work that Hadoop is a good fit for, that should be perfect: the Hadoop Task can open up ColumnFamilyStore objects on the snapshotted sstables, without having to start a full server which is nasty. Otherwise IMO we should patch Hadoop to allow Tasks to run on an existing JVM. I'm surprised HBase didn't do that: doing the copies of *all input* from one jvm to another is not insignificant. (You could take that approach w/ cassandra to, using getRangeSlice from StorageProxy started with StorageService.initClient -- actually we would want to add initLocalClient probably to mean "I only plan to query the machine I am on" -- but that would be a case of working around bad design instead of fixing it.)

27. You really want to allow arbitrary user code into your cassandra JVM? That seems like a recipe for disaster.
28. Not a whole lot worse than allowing arbitrary code into a different JVM on the same node, really. What are they going to do, read data they shouldn't? Remember we don't even have auth yet, it's very much a "power tools can maim" thing. (Note that I didn't say it should be the *only* option for Hadoop but it should definitely be *an* option.)
29. I'd be more worried about lack of memory sandboxing - they could easily OOM (which may end up killing a Cassandra thread rather than the Task), or they could create tons of objects and end up taking over the GC. I guess it could be *an* option, but it does seem awfully scary.
30. Hadoop Integration might need the following..... 1) API to return the List of splits, given the number of splits. Using this tokens we cam span equal number of MR Jobs (Have a configuration in MR Job - This will be according to the complexity in processing), which will say how many map tasks per partition and span those process. -- We have getSplit(int count) which will do it for us. 2) Start token to stream.... API Input will be Range(String startKey, Token start, Token finish, int limit).... return will be If Startwithkey is empty we will use the token1 as the starting point for the stream, else we will use startwithkey to specify the key to start with? Make sense? -- Need additional Method. So each MR jobs will get the range of data from the Cassandra and will do processing on it, it can also stream the data and doesn't need to get all of it.
31. Here's my first stab at hadoop support. I took Jeff's patches as a starting point, but the many chnages we've made to Cassandra's internals since then mean the results are pretty different. - BootUp is no longer required; instead we use the Fat Client api - Switched to ColumnFamily as the unit for InputFormat, rather than KeySpace - Use get_range_slice instead of get_key_range - Use Tokens instead of Strings for range splitting - Add build.xml and bin/ scripts for WordCount demo The combination of all this means we get RandomPartitioner support for free. We also get InputSplit location information for free. My patch 01 and 02 correspond to Jeff's 02 and 03 (no changes to Cassandra internals have been required so far). Then my 03 is just more changes to the WordCount example (I should probably squash that...) Still todo: breaking a node's range into multiple InputSplits (this will require minor changes to Cassandra) Also: as I have said before, I don't really know Hadoop, so quite possibly I did something stupid here. (For instance, Jeff's InputFormat used Writeable subclasses for both key and value; mine uses just String and ColumnFamily since that is more natural, and the IF contract does not require Writeable-ness. Is this Bad Hadoop Form?)
32. Starting the fat client itself takes a lot of time (Around 30 Sec) ... the time to sync the cluster state is unknown too. probably we need to provide a API to sync the cluster state from one of the server?
33. No. That would complicate the StorageProxy model unnecessarily. If 10s or 30s to use the StorageProxy api is too long then you should probably start working on getting Hadoop to support jobs in existing [i.e., cassandra] jvms. :) Alternatively you could write your own query parallelization code that uses the same hooks we are building Hadoop support. There's nothing magic about Hadoop per se.
34. > probably we need to provide a API to sync the cluster state from one of the server? I don't think that blocks this patch, but we can create another issue once this one is closed. It would probably be pretty easy to add a mode to the Gossiper where it was especially promiscuous, and asked a few nodes in the cluster for their full token list to get into a stable state quicker.
35. New patchset attached. Old patches 2 and 3 were squashed together as predicted, and some fixes were made to 01. New patches are 04 sub splits: split node ranges into smaller groups of keys (currently hardcoded to 4096) 03 make predicate configurable I'd like to get whatever else falls under the heading of "the least we can possibly do and be useful" done and call this ticket good, and add enhancements in future tickets.
36. * The storage-conf.xml file for contrib/word_count could probably be a diff that is applied during the build process * Why does getBootstrapToken use getRandomToken still, as opposed to midpoint? About to play around with this on a cluster... I'll let you know how it goes.
37. * The files in contrib/word_count/bin don't have the execute bit * The caller needs to include their Hadoop configuration on the classpath, * The WordCount job and all dependencies need to be wrapped in a Jar for submission * storage-conf.xml needs to be packaged in the Jar, and configurable via the classpath After applying the attached patch, I managed to get this example running on the cluster, but gossip isn't kicking in correctly yet, because the InputFormat isn't initializing the fat client.
38. it looks like your 05 squashes everything together?
39. Oops... it was late, and I was tired.
40. This patch runs initClient in the JVM that is executing the map task, and adds a check to prevent multiple initializations by the same VM. Even with this patch though, the fat client can't connect from multiple JVMs on the same machine: we have the fatClient VM using an address of 127.0.0.2, and multiple JVMs

with the same address will fail to join the gossip. The fatClient needs to be refactored to not need to listen on a port (no gossip)

41. > The fatClient needs to be refactored to not need to listen on a port (no gossip) you need to listen on a port to run messagingservice, whether or not you include gossip (remember gossip is tcp on the same port as the rest now). and w/o messagingservice you can't get responses to reads or writes. this is a non-starter. if you really need to run more than one hadoop jvm per machine you can do it by putting a storage-conf.xml on each hadoop's classpath. imo this is better than bundling it into the job jar anyway.
42. committed part of 06 as r907005. I'm working the rest of 05 and 06 into my next patchset.
43. replying to myself: > if you really need to run more than one hadoop jvm per machine it looks like the minimum task jvms you can restrict hadoop to is two (one map, one reduce), and i don't think there is a sane way to give them different classpaths. so we need to add some kind of kludge for this, short term. (long term, i think running the tasktracker and jobs inside the cassandra jvm still makes the most sense for us.)
44. Probably it is only me... i am still not comfortable in predicting the exact time when the server state is in sync with the cluster (Ghossip) it is somewhere around 10 - 30 seconds... There must be a way to predict or atleast double-check before returning the instance?
45. v6 attached. this switches to using Thrift to get range splits (and assumes CASSANDRA-775), and incorporates stu's other v5 feedback.
46. With this patch, I was able to run the job on a Hadoop cluster. * Changes to a relative tmp directory, since most users won't have access to the Hadoop /tmp * Uses the Configuration object to pass in the columnName, since the mutable static var doesn't survive distribution, * Ensures that an un-jar'd version of storage-conf.xml is first on the classpath for the benefit of the local JVM.
47. Thanks a lot! I'm +1 on getting this example in, but I still want to discuss 775.
48. committed. (But see CASSANDRA-789 for handling very large numbers of rows.)
49. Integrated in Cassandra #357 (See [http://hudson.zones.apache.org/hudson/job/Cassandra/357/])
50. We should add a CHANGES entry for this.