HadoopSupport

Contents

Overview

Hadoop integration was added way back in version 0.6 of Cassandra. It began with MapReduce support. Since then the support has matured significantly and now includes native support for Apache Pig and Apache Hive. Cassandra's Hadoop support implements the same interface as HDFS to achieve input data locality (see cluster configuration for details on data locality and how to split your analytic and realtime read loads).

DataStax, a company that creates products around Cassandra, has created a simplified way to use Hadoop with Cassandra and built it into its DataStax Enterprise product. For details on DSE, see this product page.

Top

MapReduce

Input from Cassandra

Cassandra 0.6+ adds support for retrieving data from Cassandra. This is based on implementations of InputSplit, InputFormat, and RecordReader so that Hadoop MapReduce jobs can retrieve data from Cassandra. For an example of how this works, see the contrib/word_count example in 0.6 or later. Cassandra rows or row fragments (that is, pairs of key + `SortedMap` of columns) are input to Map tasks for processing by your job, as specified by a `SlicePredicate` that describes which columns to fetch from each row.

Here's how this looks in the word_count example, which selects just one configurable columnName from each row:

```
            ConfigHelper.setColumnFamily(job.getConfiguration(), KEYSPACE, COLUMN_FAMILY);
            SlicePredicate predicate = new SlicePredicate().setColumn_names(Arrays.asList(columnName.
getBytes()));
            ConfigHelper.setSlicePredicate(job.getConfiguration(), predicate);
```

As of 0.7, configuration for Hadoop no longer resides in your job's specific storage-conf.xml. See the `README` in the `word_count` and `pig` contrib modules for more details.

Output To Cassandra

As of 0.7, there is a basic mechanism included in Cassandra for outputting data to Cassandra. The `contrib/word_count` example in 0.7 contains two reducers - one for outputting data to the filesystem and one to output data to Cassandra (default) using this new mechanism. See that example in the latest release for details.

Hadoop Streaming

Hadoop output streaming was introduced in 0.7 but was removed from 0.8 due to lack of interest and the additional complexity it added to the Hadoop integration code. To use output streaming with 0.7.x, see the contrib directory of the source download of Cassandra.

Top

Pig

Cassandra 0.6+ also adds support for Pig with its own implementation of LoadFunc. This allows Pig queries to be run against data stored in Cassandra. For an example of this, see the `contrib/pig` example in 0.6 and later.

Cassandra 0.7.4+ brings additional support in the form of a StoreFunc implementation. This allows Pig queries to output data to Cassandra. It is handled by the same class as the `LoadFunc`: `CassandraStorage`. See the `README` in `contrib/pig` for more information.

When running Pig with Cassandra + Hadoop on a cluster, be sure to follow the `README` notes in the `<cassandra_src>/contrib/pig` directory, the Cluster Configuration section on this page, and some additional notes here:

- Set the `HADOOP_HOME` environment variable to `<hadoop_dir>`, e.g. `/opt/hadoop` or `/etc/hadoop`
- Set the `PIG_CONF` environment variable to `<hadoop_dir>/conf`
- Set the `JAVA_HOME`

Pygmalion is a project created to help with using Pig with Cassandra, especially for tabular (static column names) data.

Top

Hive

Apache Hive support was done as part of a DataStax development effort. See CASSANDRA-4131] for details. There are plans to integrate this support into the Cassandra core source tree (see [https://issues.apache.org/jira/browse/CASSANDRA-4131).

Top

Oozie

Oozie, the open-source workflow engine originally from Yahoo!, can be used with Cassandra/Hadoop. Cassandra configuration information needs to go into the oozie action configuration like so:

```
<property>
    <name>cassandra.thrift.address</name>
    <value>${cassandraHost}</value>
</property>
<property>
    <name>cassandra.thrift.port</name>
    <value>${cassandraPort}</value>
</property>
<property>
    <name>cassandra.partitioner.class</name>
    <value>org.apache.cassandra.dht.RandomPartitioner</value>
</property>
<property>
    <name>cassandra.consistencylevel.read</name>
    <value>${cassandraReadConsistencyLevel}</value>
</property>
<property>
    <name>cassandra.consistencylevel.write</name>
    <value>${cassandraWriteConsistencyLevel}</value>
</property>
<property>
    <name>cassandra.range.batch.size</name>
    <value>${cassandraRangeBatchSize}</value>
</property>
```

Note that with Oozie you can specify values outright like the partitioner here, or via variable that is typically found in the properties file. One other item of note is that Oozie assumes that it can detect a filemarker for successful completion of the job. This means that when writing to Cassandra with, for example, Pig, the Pig script will succeed but the Oozie job that called it will fail because filemarkers aren't written to Cassandra. So when you write to Cassandra with Hadoop, specify this property to avoid that check. Oozie will still get completion updates from a callback from the job tracker, but it just won't look for the filemarker.

```
<property>
    <name>mapreduce.fileoutputcommitter.marksuccessfuljobs</name>
    <value>false</value>
</property>
```

Top

Cluster Configuration

If you would like to configure a Cassandra cluster yourself so that Hadoop may operate over its data, it's best to overlay a Hadoop cluster over your Cassandra nodes. You'll want to have a separate server for your Hadoop `NameNode/JobTracker`. Then install a Hadoop `TaskTracker` on each of your Cassandra nodes. That will allow the `JobTracker` to assign tasks to the Cassandra nodes that contain data for those tasks. Also install a Hadoop `DataNo` de on each Cassandra node. Hadoop requires a distributed filesystem for copying dependency jars, static data, and intermediate results to be stored.

The nice thing about having a `TaskTracker` on every node is that you get data locality and your analytics engine scales with your data. You also never need to shuttle around your data once you've performed analytics on it - you simply output to Cassandra and you are able to access that data with high random-read performance. Note that Cassandra implements the same interface as HDFS to achieve data locality.

A note on speculative execution: you may want to disable speculative execution for your hadoop jobs that either read or write to Cassandra. This isn't required, but may be helpful to reduce unnecessary load.

One configuration note on getting the task trackers to be able to perform queries over Cassandra: you'll want to update your `HADOOP_CLASSPATH` in your `<hadoop>/conf/hadoop-env.sh` to include the Cassandra lib libraries. For example you'll want to do something like this in the `hadoop-env.sh` on each of your task trackers:

```
export HADOOP_CLASSPATH=/opt/cassandra/lib/*:$HADOOP_CLASSPATH
```

Virtual Datacenter

One thing that many have asked about is whether Cassandra with Hadoop will be usable from a random access perspective. For example, you may need to use Cassandra for serving web latency requests. You may also need to run analytics over your data. In Cassandra 0.7+ there is the NetworkTopologyStrategy which allows you to customize your cluster's replication strategy by datacenter. What you can do with this is create a 'virtual datacenter' to separate nodes that serve data with high random-read performance from nodes that are meant to be used for analytics. You need to have a snitch configured with your topology and then according to the datacenters defined there (either explicitly or implicitly), you can indicate how many replicas you would like in each datacenter. You would install task trackers on nodes in your analytics section and make sure that a replica is written to that 'datacenter' in your NetworkTopologyStrategy configuration. The practical upshot of this is your analytics nodes always have current data and your high random-read performance nodes always serve data with predictable performance.

Top

Troubleshooting

If you are running into timeout exceptions, you might need to tweak one or both of these settings:

- Each input split is divided into sequential batches of rows requested at a time from Cassandra. This is the **cassandra.range.batch.size** property and it defaults to 4096. If you are experiencing timeouts, you might first try to reduce the batch size so that it can more easily complete the request within the timeout. This is either specified in your hadoop configuration or using `org.apache.cassandra.hadoop.ConfigHelper.setRangeBatchSize`.
- Starting in Cassandra 1.2, there is range request specific timeout called **range_request_timeout_in_ms** in the cassandra.yaml. Hadoop requests data in sequential batches and each request has to complete within this timeout. Prior to Cassandra 1.2, you're can set the general **rpc_timeout_in_ms** higher, which affects timeouts for reads, writes, and truncate operations in addition to range requests.

If you still see timeout exceptions with resultant failed jobs and/or blacklisted tasktrackers, there are settings that can give Cassandra more latitude before failing the jobs. An example of usage (in either the job configuration or tasktracker mapred-site.xml):

```
<property>
  <name>mapred.max.tracker.failures</name>
  <value>20</value>
</property>
<property>
  <name>mapred.map.max.attempts</name>
  <value>20</value>
</property>
<property>
  <name>mapred.reduce.max.attempts</name>
  <value>20</value>
</property>
```

The settings normally default to 4 each, but some find that too conservative. If you set it too low, you might have blacklisted tasktrackers and failed jobs because of occasional timeout exceptions. If you set them too high, jobs that would otherwise fail quickly take a long time to fail, sacrificing efficiency. Keep in mind that this can just cover a problem. It may be that you always want these settings to be higher when operating against Cassandra. However, if you run into these exceptions too frequently, there may be a problem with your Cassandra or Hadoop configuration.

If you are seeing inconsistent data coming back, consider the consistency level at which you read (**cassandra.consistencylevel.read**) and write (**cassandra.consistencylevel.write**). Both properties default to ConsistencyLevel.LOCAL_ONE (Previously ONE).

Also hadoop integration uses range scans underneath which do not do read repair. However reading at ConsistencyLevel.QUORUM will reconcile differences among nodes read. See ReadRepair section as well as the ConsistencyLevel section of the API page for more details.

If you're having trouble with a never ending supply of data when using `ColumnFamilyInputFormat`, make sure you're not altering the `ByteBuffer` key input to the mapper; either by duplicating it first or using one of the absolute accessors.

Top

Support

Sometimes configuration and integration can get tricky. To get support for this functionality, start with the `examples` directory in the source download of Cassandra. Make sure you are following instructions in the `README` file for that example. You can search the Cassandra user mailing list or post on there as it is very active. You can also ask in the #Cassandra irc channel on freenode for help. Other channels that might be of use are #hadoop, #hadoop-pig, and #hive. Those projects' mailing lists are also very active.

There are professional support options for Cassandra that can help you get everything working together. For more information, see ThirdPartySupport. There are also professional support options specifically for Hadoop. For more information on that, see Hadoop's third party support wiki page.

Top