Wiki Operacional PSafe

Hadoop Good Practices

Caching

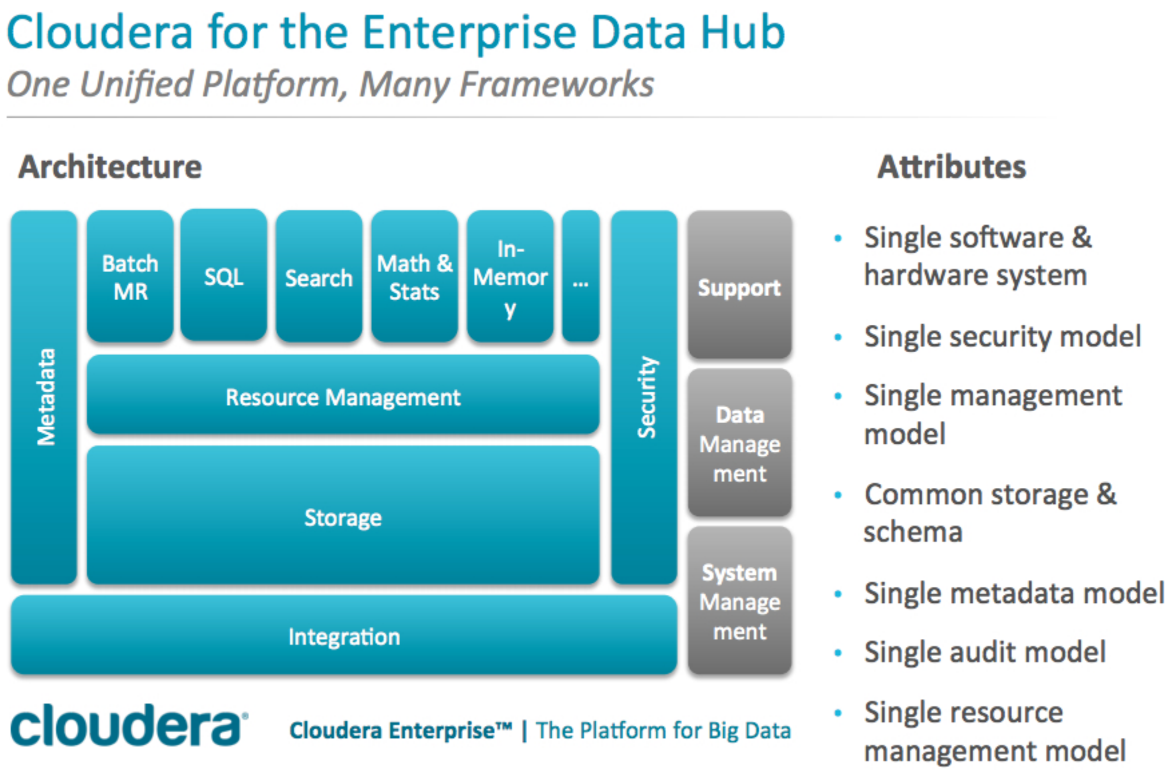
Centralized cache management in HDFS is an explicit caching mechanism that allows users to specify paths to be cached by HDFS. The NameNode will communicate with DataNodes that have the desired blocks on disk, and instruct them to cache the blocks in off-heap caches.

This has several advantages:

- Explicit pinning prevents frequently used data from being evicted from memory. This is particularly important when the size of the working set exceeds the size of main memory, which is common for many HDFS workloads.
- Since DataNode caches are managed by the NameNode, applications can query the set of cached block locations when making task placement decisions. Co-locating a task with a cached block replica improves read performance.
- When block has been cached by a DataNode, clients can use a new, more-efficient, zero-copy read API. Since checksum verification of cached data is done once by the DataNode, clients can incur essentially zero overhead when using this new API.
- Centralized caching can improve overall cluster memory utilization. When relying on the OS buffer cache at each DataNode, repeated reads of a block will result in all n replicas of the block being pulled into buffer cache. With centralized cache management, a user can explicitly pin only m of the n replicas, saving n-m memory.

http://www.cloudera.com/content/cloudera/en/documentation/cdh5/v5-0-0/CDH5-Installation-Guide/cdh5ig_hdfs_caching.html
[http://www.cloudera.com/content/cloudera/en/documentation/cdh5/v5-0-0/CDH5-Installation-Guide/cdh5ig_hdfs_caching.html]

Data Hub



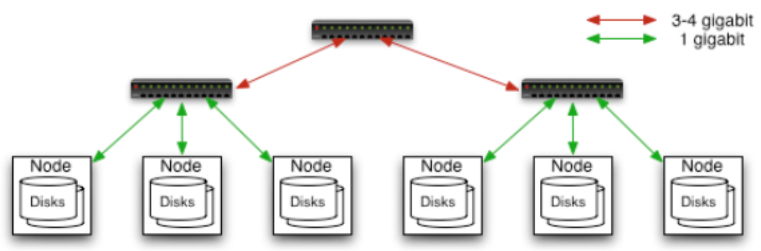
Hadoop File Types

There are several Hadoop specific file formats that were specifically created to work well with MapReduce. These Hadoop specific file formats include file based data structures such as sequence files, serialization formats like Avro, and columnar formats such as RCFiles and Parquet.

These file formats have differing strengths and weaknesses, but all share the following characteristics that are important for Hadoop applications:

- Splittable compression: these formats support common compression formats and are also splittable. We'll discuss splittability more in the section on compression, but note that the ability to split files can be a key consideration when storing data in Hadoop, since it allows large files to be split for input to MapReduce and other types of jobs. The ability to split a file for processing by multiple tasks is of course a fundamental part of parallel processing, and is also key to leveraging Hadoop's data locality feature.
- Agnostic compression: the file can be compressed with any compression codec, without readers having to know the codec. This is p

Network



Typically in 2 level architecture
– Nodes are commodity PCs
– 30-40 nodes/rack
– Uplink from rack is 3-4 gigabit
– Rack-internal is 1 gigabit

Parquet

Impala helps you to create, manage, and query Parquet tables. Parquet is a column-oriented binary file format intended to be highly efficient for the types of large-scale queries that Impala is best at. Parquet is especially good for queries scanning particular columns within a table, for example to query "wide" tables with many columns, or to perform aggregation operations such as SUM() and AVG() that need to process most or all of the values from a column. Each data file contains the values for a set of rows (the "row group"). Within a data file, the values from each column are organized so that they are all adjacent, enabling good compression for the values from that column. Queries against a Parquet table can retrieve and analyze these values from any column quickly and with minimal I/O.

http://www.cloudera.com/content/cloudera/en/documentation/cloudera-impala/latest/topics/impala_parquet.html [http://www.cloudera.com/content/cloudera/en/documentation/cloudera-impala/latest/topics/impala_parquet.html]

JobHistoryServer

Requires 8 GB of memory.

Resource Manager

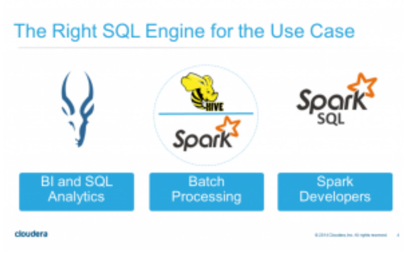
Requires 4 to 8 GB of memory.

High Availability with Zookeeper: automatic failover.

It is possible to start multiple Resource Managers. Need to associate hostnames and rm-id's. Web-Uis automatically get redirected to the active.

SQL in Hadoop

Cloudera's good practices:


The Right SQL Engine for the Use Case

BI and SQL Analytics

Batch Processing

Spark Developers

cloudera

YARN

We need to configure YARN resource manager in order to use full power of our hardware while running map reduce job.

Here is the recommendation based on our datanodes hardware configuration:

- yarn.nodemanager.resource.memory-mb = 32 GB

- yarn.nodemanager.resource.cpu-vcores = 10
- mapreduce.[map|reduce].cpu.vcores = 1
- map.memory.mb = 2 GB
- reduce.memory.mb = 3 GB
- mapreduce.map.java.opts.max.heap = 1650 MB (could be configured in job)
- mapreduce.reudce.java.opts.max.heap = 2611 MB (could be configured in job)

Note: Heap size (mapreduce.[map|reduce].java.opts.max.heap should be less than RM config ([map|reduce].memory.mb) This will allow us to limit map reduce jobs to 20 GB of RAM and 8 cores per node. The rest could be used by other services like HBase.

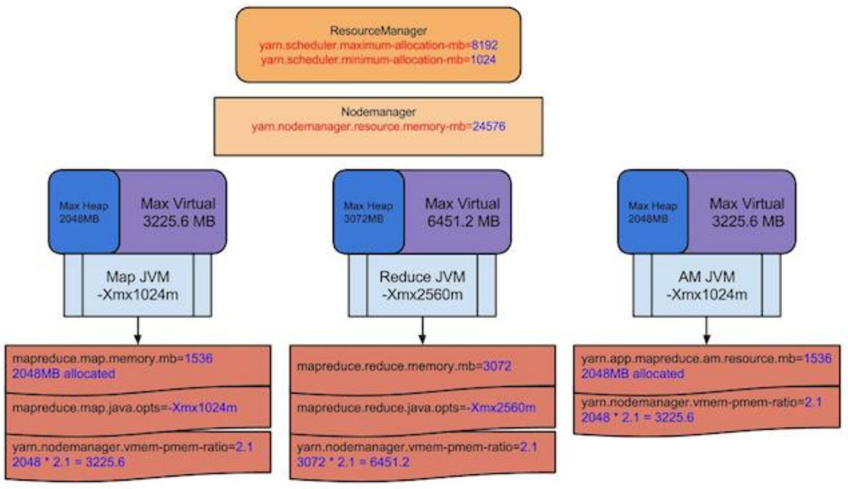
With the configuration above RM will allocate for each map task:

- 2 GB of physical RAM
- 4.2 GB upper limit of virtual memory (with default ratio of 2.1)
- 1.6 GB upper limit of JVM heap

and reduce task:

- 3 GB of physical RAM
- 6.3 GB upper limit of virtual memory (with default ratio of 2.1)
- 2.6 GB upper limit of JVM heap

If you want an explanation of how does YARN resource manager allocates resource, below is a good answer on Quora: http://www.quora.com/How-do-I-configure-the-number-of-reducers-in-YARN [http://www.quora.com/How-do-I-configure-the-number-of-reducers-in-YARN]


ResourceManager
yarn.scheduler.maximum-allocation-mb=8192
yarn.scheduler.minimum-allocation-mb=1024
Nodemanager
yarn.nodemanager.resource.memory-mb=24576
Max Heap 2048MB
Max Virtual 3225.6 MB
Map JVM -Xmx1024m
Max Heap 3072MB
Max Virtual 6451.2 MB
Reduce JVM -Xmx2560m
Max Heap 2048MB
Max Virtual 3225.6 MB
AM JVM -Xmx1024m
mapreduce.map.memory.mb=1536
2048MB allocated
mapreduce.map.java.opts=-Xmx1024m
yarn.nodemanager.vmem-pmem-ratio=2.1
2048 * 2.1 = 3225.6
mapreduce.reduce.memory.mb=3072
mapreduce.reduce.java.opts=-Xmx2560m
yarn.nodemanager.vmem-pmem-ratio=2.1
3072 * 2.1 = 6451.2
yarn.app.mapreduce.am.resource.mb=1536
2048MB allocated
yarn.nodemanager.vmem-pmem-ratio=2.1
2048 * 2.1 = 3225.6

Resource: https://support.pivotal.io/hc/en-us/articles/201462036-Mapreduce-YARN-Memory-Parameters [https://support.pivotal.io/hc/en-us/articles/201462036-Mapreduce-YARN-Memory-Parameters]

/var/www/html/dokuwiki/data/pages/documentacao/bi/hadoop_good_practices.txt · Última modificação: 12/07/2015 13:09 por Mikhail Bragin