Hadoop cluster performance tuning

**Best performance for Hadoop cluster**

**How to ensure best performance for Hadoop Cluster?**

* Installing Hadoop cluster in production is just half the battle won. It is extremely important for a Hadoop admin to tune the Hadoop cluster setup to gain maximum performance. During Hadoop installation, the cluster is configured with default configuration settings which are on par with the minimal hardware configuration. It is extremely important for Hadoop admins to be familiar with various hardware specifications like – the number of disks mounted on datanodes, RAM capacity, the number of virtual and physical cores, the number of CPU cores, NIC Cards, etc.
* There is no single performance tuning technique that can fit all hadoop jobs because it is very difficult to obtain equilibrium among the various resources whilst solving the big data problem. The performance tuning tips and tricks vary based on the amount of data that is being moved and also on the type of Hadoop job being run in production. This document highlights some of the best performance tuning tips for Hadoop jobs to achieve maximum performance.
* The biggest selling point for Apache Hadoop driving enterprise adoption, as a big data processing framework is - the cost effectiveness in setting up data centers for processing huge amounts of structured and unstructured data. However, the major roadblock in obtaining maximum performance from a hadoop cluster is its core hardware stack.  Considering commodity hardware as the major thing, it is extremely necessary for a Hadoop admin to make the best use of a Hadoop cluster’s capability to achieve best performance from the hardware stack. Hadoop being horizontally scalable, most of the hadoop administrators continue adding nodes or instances into a hadoop cluster to enhance the performance. However, doing so leads to massive hadoop clusters which do not run on optimal configurations leading to huge operational costs.

Let’s explore some of the best and most effective performance tuning techniques, to set up hadoop clusters in production with commodity hardware, to enhance performance with minimal operational cost:

**1) Memory Tuning**

* The foremost step to ensure maximum performance for a Hadoop job, is to tune the best configuration parameters for memory, by monitoring the memory usage on the server. Apache Hadoop has various options on memory, disk, CPU and network that helps optimize the performance of the hadoop cluster. Every Hadoop MapReduce job collects information about various input records read, number of records pipelined for further execution, number of reducer records, heap size set, swap memory, etc. Generally, hadoop tasks are not bounded by CPU- the prime concern should be to optimize the memory usage and disk spills.
* A thumb rule when tuning the memory is to ensure that the jobs don’t trigger swapping. Swap memory usage can be monitored using software like Ganglia, Nagios or Cloudera Manager. Whenever there is excess swap memory utilization, memory usage should be optimized by configuring the *mapred.child.java.opts* property by reducing the amount of RAM that is allotted to each task in *mapred.child.java.opts.* The memory for the task can be adjusted by setting the mapred.child.java.opts to -Xmx2048M in the mapred-site.xml file as shown below-

    mapred.child.java.opts

    -Xms1024M -Xmx2048M

**2) Improving IO Performance**

Here are some key points to be followed to optimize the MapReduce performance by ensuring that the Hadoop cluster configuration is tuned-

* Linux OS has a checkpoint for each file including checksum, last accessed time, creation time, user who created the file, etc. To achieve better IO performance, the checkpoint should be disabled in HDFS - as HDFS supports write-once-read-many times’ model. The applications will be able to access the data on HDFS in a random fashion.
* The mount points for DataNode or data directories should be configured with the noatime option to ensure that the metadata is not updated by the NameNode every time the data is accessed. The mounts for MapReduce storage and DFS, when mounted with noatime option, deactivates access time tracking - rendering enhanced IO performance.
* The two configuration parameters ‘mapreduce.local.dir’ and ‘dfs.data.dir’ should be set such that they point to a one directory on each of the disks. This helps make the best use of the overall IO capacity.

It is recommended not to use LVM and RAID on DataNode or TaskTracker machines as it reduces performance.

**3) Minimizing the Disk Spill by Compressing Map Output**

Disk IO is one of the major performance bottleneck and here are two ways that help minimize disk spilling:

1. Ensure that the mapper for your MapReduce job uses 70% of heap memory for spill buffer.
2. Compress Mapper Output

* **LZO Compression**
* When the Map Output is very large, intermediate data size should be reduced using various compression techniques like LZO, BZIP, Snappy, etc. Map Output is not compressed by default and to enable compression of Map Output - mapreduce.map.output.compress should be set to true. ‘mapreduce.map.output.compress.codec’ should be set based on whatever compression technique is used LZO, BZIP or Snappy.
* Every MapReduce job that produces large Map Output is likely to benefit from intermediate data compression with LZO. It initially might seem to be an overhead for the CPU but minimal number of disk IO operations during the shuffle phase will considerably improve the performance. With LZO compression every 1GB of output data save approximately 3GB of disk writes.
* If a huge amount of data is being written to the disk during execution of the Map tasks - then increasing the memory size of the buffer helps. Generally, when the map task is not able to hold the data into the memory it spills it to local disk which is a time consuming process because of the number of IO operations involved. To avoid this situation, configure the parameters *io.sort.mb and* *io.sort.factor to increase the size of buffer memory and attain optimal*perform*ance.*

4) Tuning the Number of Mapper or Reducer Tasks

Every map or reduce task usually takes 40 seconds to complete execution. Also when there is a large job to be executed it does not make the best use of all the slots in the available cluster. Thus, it is extremely important to tune the number of map or reduce tasks using the following techniques-

* If the MapReduce job has more than 1 terabyte of input, then, to ensure that the number of tasks are smaller- the block size of the input dataset should be increased to 512M or 256M.  The block size of existing files can be modified by configuring the dfs.block.size property. Once the command to change the block size is executed the original data can be removed.
* If the MapReduce job on the hadoop cluster launches several map tasks wherein each task completes in just few seconds -then reducing the number of maps being launched for that application without impacting the configuration of the hadoop cluster will help optimize performance. This ensures that the task load on the application master is reduced whilst allocating the desired resources.
* Setting up and scheduling tasks requires time overhead. If a task takes less than 30 seconds to execute, then it is better to reduce the number of tasks. Reusing JVM is a good alternative to this problem.

5) Writing a Combiner

Based on the Hadoop environment, apart from data compression technique, writing a combiner to reduce the amount of data to be transferred can also prove to be beneficial. If the job performs a large shuffle wherein the map output is several GBs per node or if the job performs an aggregation sort – writing a combiner can help optimize the performance. Combiner acts as an optimizer for the MapReduce job. It runs on the output of the Map phase to reduce the number of intermediate keys being passed to the reducers. This reduces the load of the reduce task in processing the business logic.

6)  Using Skewed Joins

Using standard joins in the transformation logic with Pig or Hive tools can at times result in weird performance of the MapReduce jobs, as the data being processed might have some skewness - meaning 80% of the data is going to a single reducer. If there is a huge amount of data for a single key, then one of the reducer will be held up with processing majority of the data –this is when Skewed join comes to the rescue. Skewed join computes a histogram to find out which key is dominant and then data is split based on its various reducers to achieve optimal performance.

7)  Speculative Execution

·The performance of MapReduce jobs is seriously impacted when tasks take a long time to finish execution. Speculative execution is a common approach to solve this problem by backing up slow tasks on alternate machines. Setting the configuration parameters ‘mapreduce.*map.tasks.speculative.execution’ and ‘mapreduce.reduce.tasks.speculative.execution’* to true will enable speculative execution so that the job execution time is reduced if the task progress is slow due to memory unavailability.

There are several performance tuning tips and tricks for a Hadoop Cluster and we have highlighted some of the important ones.