**MR Performance Tuning**

1. Max input splits - We can set with a configuration mapred.max.split.size as bytesize

If you want to process 1gb file with 2 mappers you can set conf.set(mapred.max.split.size, 536870912)

2. mapred.job.reuse.jvm.num.tasks this property can be set for reusing the jvm of an existing container task so rather than launching new containers it will reuse but this is helpful only if you have small tasks if the task is a long running one then this is not useful, this property can be working only in MR version1, YARN will not support this.

3. mapreduce.tasktracker.map.tasks.maximum - The maximum number of map tasks that will be run simultaneously by a task tracker.

mapreduce.tasktracker.reduce.tasks.maximum - The maximum number of reduce tasks that will be run simultaneously by a task tracker.

4. (This is applied for YARN -MR also) - mapred.output.compress: This boolean flag will define is the whole map/reduce job will output compressed data. I would always set this to true also. Faster read/write speeds and less disk spaced used.

5. Invoke combiner - If you set the setcombinerclass(reducer.class) then your mapper will perform local reducer activity hence the shuffle bytes of data will be reduced, this property can be used only for the case of commutative and associative data sets.

6. max in memory buffer size -

mapreduce.reduce.shuffle.input.buffer.percent tells about the percentage of the reducer's heap memory to be allocated for the circular buffer to store the intermediate outputs copied from multiple mappers.

mapreduce.reduce.shuffle.memory.limit.percent tells about the maximum percentage of the above memory buffer that a single shuffle (output copied from single Map task) should take.

mapreduce.reduce.shuffle.merge.percent tells about the threshold percentage by where the in-memory merger thread will run to merge the available shuffle contents on the memory buffer into a single file and immediately spills the merged file into the disk.

7. skip bad records - Skipping mode is off by default; you enable it independently for map and reduce tasks using the SkipBadRecords class. It’s important to note that skipping mode can detect only one bad record per task attempt, so this mechanism is appropriate only for detecting occasional bad records (a few per task, say). You may need to increase the maximum number of task attempts (via mapred.map.max.attempts and mapred.reduce.max.attempts

8. A small file is one which is significantly smaller than the HDFS block size (default 128MB). If you’re storing small files, then you probably have lots of them, using a custom CombineFileInputFormat for benchmarking can speedup the program from 3 hours to 23 minutes, and after some further tuning, the same task can be run in 6 minutes!

9. Speculative exec - In MapReduce, a job is divided into tasks and executed in parallel across the cluster , which is faster compared to sequential execution.

In this parallel processing, there might be a chance for some/few of the tasks in some nodes to be slow compared to other tasks (in the other nodes) of the same job.

The reason can be anything: node busy, network congestion, etc, which limits the total execution time of the Job, and the system should wait for the slow running tasks to be completed. Here, Hadoop doesn't try to fix the slow running tasks, instead clones the slow running tasks to the other nodes, where the rest of the tasks are completed, and gets executed there. This is termed as Speculative execution .

10. Parallel copying in reducer side.

mapreduce.reduce.shuffle.parallelcopies: Number of threads used to copy map outputs to reducers.