29th October, 2018

Map-Reduce Comparison Updates Report

D Swami

Project Title: On studying the performance of Hadoop Map Reduce vs MPI for Aggregation Operations: A Big Data Challenge

The following work introduces the results of my analysis for Map Reduce Hadoop workload. Using 200 GB of data from NYC TLC taxi trip records. The analysis time is as follows:

1) Block Size: Keeping the input split size constant at 64 MB, uber mode disabled and Snappy compression format, we found that larger block-size performs better than the lower value when using Wall clock time as a parameter as shown in Figure 1. Although, I have no conclusive theory but it's clear that larger block size takes less time to compute mapper operation.

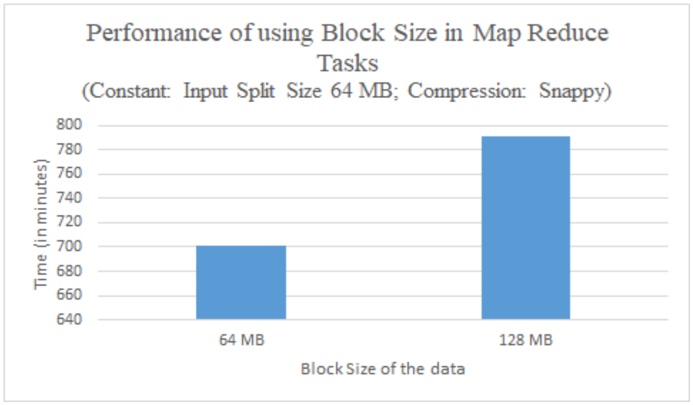

Figure 1: Wall Clock Time when using different Block size in Map Reduce Hadoop (keeping uber mode disabled, input split size at 64 MB and using snappy compression)

2) Input-Split Size: Keeping the block size constant at 128 MB, uber mode disabled, and snappy compression format. We have conclusive evidence that larger Input-split size performs better as shown in Figure 2. The reason being the fact that overload associated with spinning a new mapper function is really high. For instance average time to execute a mapper on 64 MB Input split size vs 128 MB split size were 14 second and 16 second respectively. Now given than a difference of 2 second can process 64 MB of data more, it makes sense as to why a higher input-split size is better. However, it's been recommended not to keep input split-size greater than block size on official Apache Hadoop website.

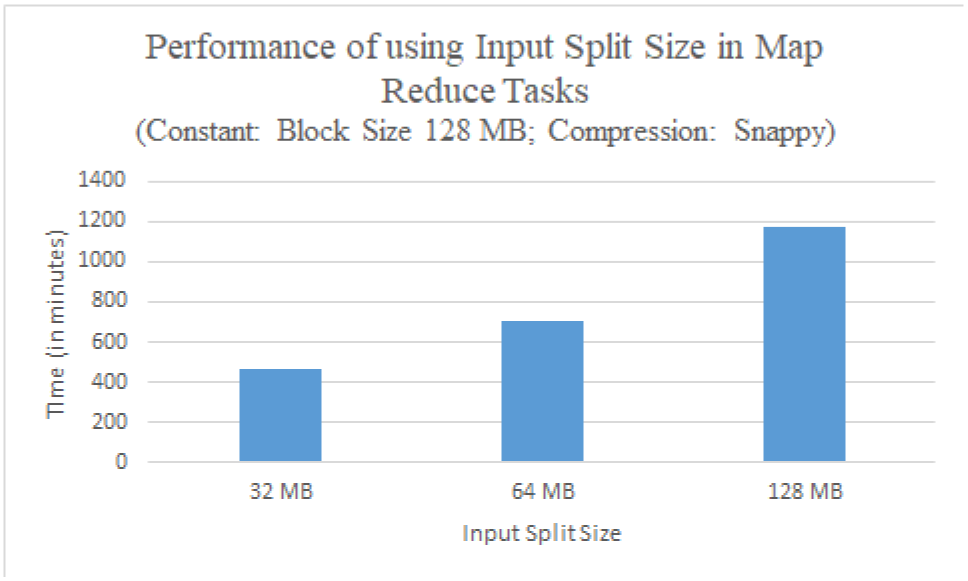

Figure 2: Wall Clock Time when using different Input Split size in Map Reduce Hadoop (keeping uber mode disabled, block size of 128 MB and using snappy compression)

3) Compression Format: Keeping the Input split size constant at 128 MB, uber mode disabled, and block size of 128 MB. It is found that use of snappy compression helps reduce processing time during the following stage of Map-Reduce Framework: "Shuffle", "Merge", and "Reduce". The comparison of the timing for these is presented in Table 1. Thus, compression helps reduce task completion time in Map Reduce. Results are presented in Figure 3.

Table 1 Comparison to wall clock time taken by different Map Reduce Stages when using Compression vs No-Compression in Map Reduce

Map Reduce Stage	Timing W/O Compression	Timing using Snappy Compression
Shuffle	14 min 27 s	9 min 26 s
Merge	16 min 35 s	4 min 43 s
Reduce	19 min 07 s	7 min 24 s

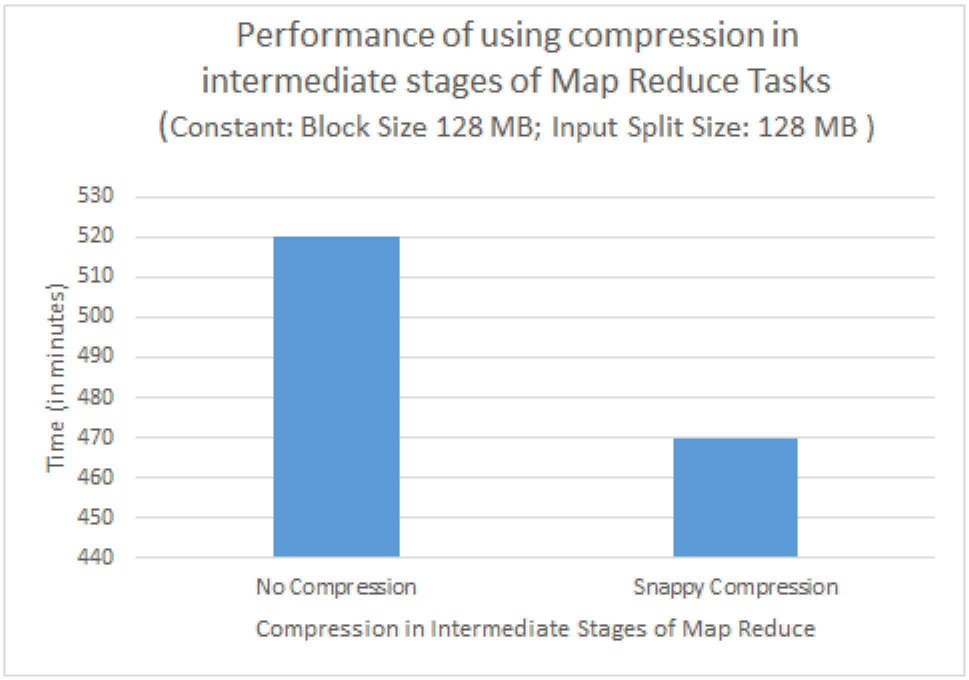

Figure 3: Wall Clock Time when using compression vs no compression in Map Reduce Hadoop (keeping uber mode disabled, input split size at 128 MB and using block size of 128 MB)

PS (More on Uber kept disabled): *Uber mode disabled actually means that I am keeping Mapper and reducer as separate processes. Keeping this mode on would launch Mapper and Reducers inside the Application master. The consequence of which I am not sure off. The limited information I have suggests that it's helpful to enable uber mode while running an interactive query (say using hive or other databases atop Hadoop). For Map Reduce, it had been encouraged if you had no more than one reducer and/or you have map only jobs (This information is 4 years old and may not even hold true.). More recently, a blog on claimed that "For running a job as uber task in YARN, job has to be 'sufficiently small'". However, the term sufficiently small has not even been loosely define and hence remains unclear if using uber mode is a good idea or not. As of now, I have decided in favor of not pursuing this feature and also I have not claimed the same in my proposal. So it seems fair to me to be able to say that keeping uberized mode disabled I have following results.*