A Framework for Performance Analysis and Tuning in Hadoop Based Clusters

Garvit Bansal
Anshul Gupta
Utkarsh Pyne
LNMIIT, Jaipur, India
Email: [garvit.bansal
anshul.gupta
utkarsh.pyne] @lnmiit.ac.in

Manish Singhal
IIT - BHU, Varanasi, India
Email: manish.singhal.ece11@itbhu.ac.in

Subhasis Banerjee
IIIT Delhi, India
Email: subhasis@iiitd.ac.in

***Abstract*—Big Data computing platforms such as MapReduce frameworks is foraying into the domain of high performance computing with stringent non-functional requirements namely execution times and throughputs. Over the last couple of years, several hundreds of sequential programs in various domains like biological informatics, health-care and financial domains have been converted into parallel paradigms. Movement of such time sensitive application will harden the problem of optimal resource utilization on the MapReduce frameworks. Traditional scheduling have been predominantly handling similar workflow with pre-defined non-functional requirements on diverse set of resources. Thanks to the Hadoop which provides us with flexibility of varying various parameters according to our choice but this facility proves to be the main bottleneck as configuring too many parameters with a perfect balance between all of them to get the best result is a time consuming and a challenging job. In our work,we attempt to analyze the effect of various configuration parameters on Hadoop Map-Reduce performance under various conditions, to achieve maximum throughput. Using these methodologies we have been able to achieve performance improvements. We study through extensive experiments, the impact of various configuration parameters and suggest an optimal value in each case.**

I. Introduction

Parallel computing Platform such as Hadoop that implement the MapReduce framework have become defacto platform for Big Data Computing. Such platform have shown in the past to significantly help the class of applications called embarrassingly parallel programs. MapReduce have been extended to several domains like bioinformatics (CloudBurst), Big-Data retention analytics from IBM, advertisement campaigning, financial sectors and many more [6], [7], [8]. This increasingly diverse set of domain applications getting on-boarded to this platform especially brings in additional set of concerns. Applications especially those in finance, customer analytics bring in the baggage of diverse and stringent set of non-functional parameter requirements. For example, every click in ad campaigning is associated with handling large, complex data volumes requires sub-millisecond latency to make optimized decisions on real time ad placement. Unlike deploying Hadoop clusters and implementing Hadoop applications, tuning Hadoop clusters for performance is not a well documented and widely-understood area.

In this paper we focus on two key performance indicators viz., throughput and execution time. It should be noted that these key performance are governed by data placement characteristics which in-turn governs the scheduling of job that act on these data. Several parameters control the performance of data placement. Block placement is influenced by block sizes, cache available in the individual machines, spill memory used etc.In this paper, we perform extensive experiments on a well established and widely used distributed computing platform viz, Hadoop cluster and study throughly the impact of the data placement and scheduling algorithms on the non-functional parameters. We show that providing a simple yet sophisticated data placement that considers several aspects of the computing platform (such as cache size, memory, spill memory etc.,) and the nature of the jobs submitted can increase the throughput of the jobs completed by several orders.

Our paper is organized as follows. In section II we present the Setup of our experiments. We describe our experimental environment and results of our approach

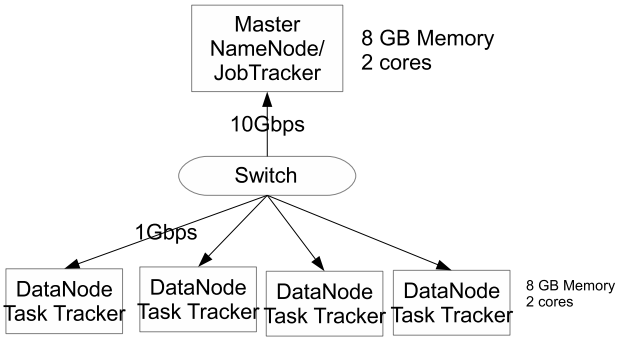

Fig. 1. Experimental Hadoop Cluster

in section III. A framework for performance tuning of Hadoop cluster is given in section IV. We briefly describe prior work in section V and conclude our work in section VI with an outline of possible future work.

II. Experimental Setup

We evaluate the performance of our system and scheduling algorithm on a Hadoop Cluster (Hadoop version 1.1.2) with a single master node and 4 slave machines. Each slave machine consists of 2 core. A dedicated switch of 10 Gbps uplink and the 1 Gbps lateral ports were used for the connectivity of the machine. Figure 1 shows the different networking and the computing parameters of the setup. Purdue MapReduce Benchmark Suite (PUMA) is used to evaluate the performance of our system [1]. Our focus is mainly on two of the key performance metrics in cluster based systems ie., Execution time (both overall and average) and throughput (number of jobs completed per unit of time).

We assume the arrival of the jobs to follow a Poisson distribution and simulate the arrival of jobs for different mean rates (λ=0.5, 0.2, 0.7). We choose through a uniform distribution a job that needs to be schedule at each arrival time from the bucket of the PUMA benchmarks. In other words, each job that is deemed to be scheduled is selected from one of the PUMA benchmarks through a uniform distribution. We observe through this process around 70-90 jobs (uniformly picked up from the PUMA set) that can be simulated to arrive in about 60 min of time units. Furthermore, we experimented with different sizes of input data starting from 1 GB upto 8 GB of data.

III. Experiments and Results

In this section, we present our analysis on the experiments performed on the Hadoop cluster. In each experiment, we compare three scheduling algorithms namely FIFO, Fair sharing and Capacity scheduling. To reemphasize, our focus is on studying the effect of different data block characteristics and their placement on the overall execution time and throughput. Our study also derives hints from the existing set of work on fine tuning the Hadoop cluster for maximum performance

A. Effect of different block sizes

We vary the block sizes of the data that is used by the HDFS file system for data distribution. Block sizes have the following effects on the performance metrics. Higher the block sizes it is easier for the HDFS to manage the information in the Namenode and lesser is the communication to the Namenode. Furthermore, when multiple set of data blocks for different applications are placed on a data node, accesses to the I/O (through I/O scheduling) by different applications will have an impact on the completion times. On the other hand, increasing the block size reduces the parallelism that can be exploited across the clusters. When the cluster size is large, this will have a huge impact on the overall execution time.

Figure 2 shows the effect of block sizes on overall execution and average execution time. Two observations can be inferred from these graphs: (a) Data block sizes have a huge by indeterministic effect on the execution times. (b) Choice of scheduler seems to have significant impact on the execution times. The capacity scheduler seems to be performing poorly compared to the FIFO and Fair scheduler. FIFO yields better on average execution time than the Fair Scheduler. This is due to fact that individual units of jobs and get complete set of resources and for the same data input size there the execution time do not change. The fair scheduler performs better in the case of overall execution time since each job irrespective of the length of the data performs gets equal time slices for execution. This shows that the choice of the scheduler and the block sizes play an key role in the performance of the Hadoop clusters.

B. Effect of copy phase

The copy phase/shuffle phase is performed when the MAP jobs complete their execution and their output is available to be transferred to the reducers. The reducers (set of machines to execute the reduce jobs) are decided based on the key output by the map programs. The operation is expensive in terms of the time consumed

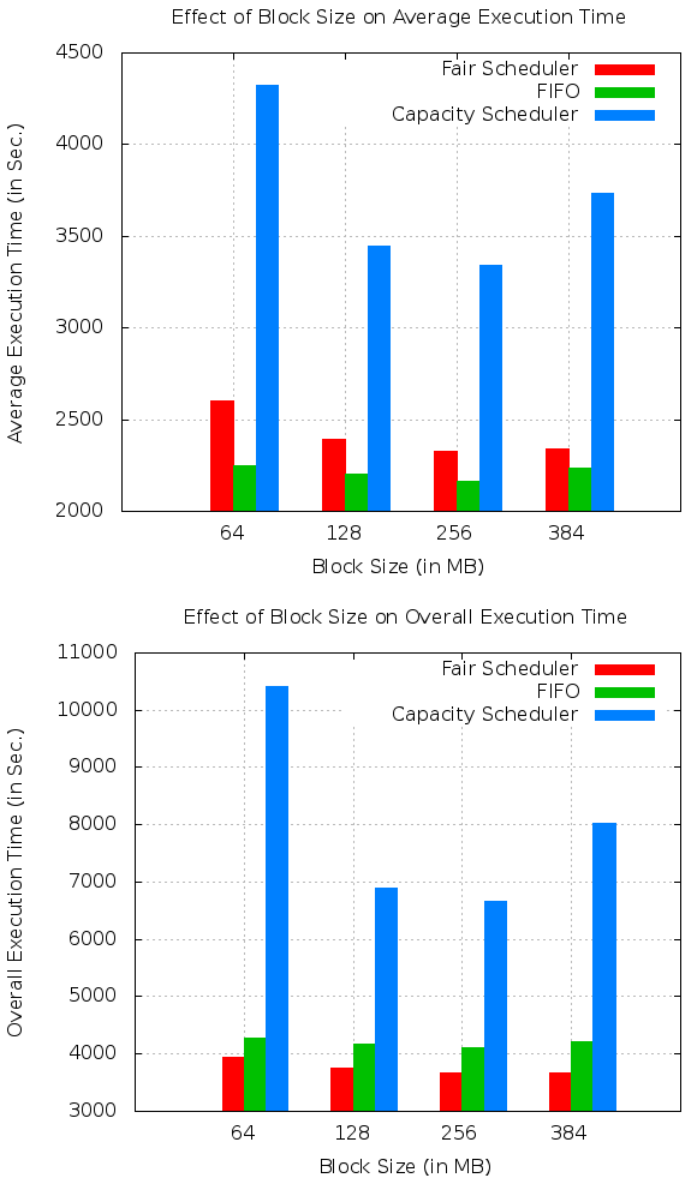

Fig. 2. Effect of varying block sizes for distribution and different scheduling algorithms

as it involves waiting for the data to be available for the jobs and also network to be relatively free to transfer the data at line speed. Effect of copy phase can be shown in fig. 3 for FIFO and Fair scheduler. It can be observed that the execution time (overall) varies significantly on the block sizes and the number of threads used to read the data from the output of the MAP jobs. While increasing the number of threads in lower block sizes (128 MB) reduces the execution time (since the more the number of threads, more is the data read in small amounts), it does not reduce by similar factor using a larger block size (256 MB). But the execute time is lesser using 256 MB compared to 128 MB and 384 MB. Among the results for 256 MB, lower number of thread in FIFO yields better execution times due to the fact that increasing the threads results in queues and contentions within the I/O for reading data blocks.While in Fair Scheduling increasing the threads results in better execution time. High block sizes lose on the parallelism that can be exploited and do not yield good performance metrics. Thus we observe that thread sizes for the number of copy operations is also one of the main factors that effects the performance metrics.

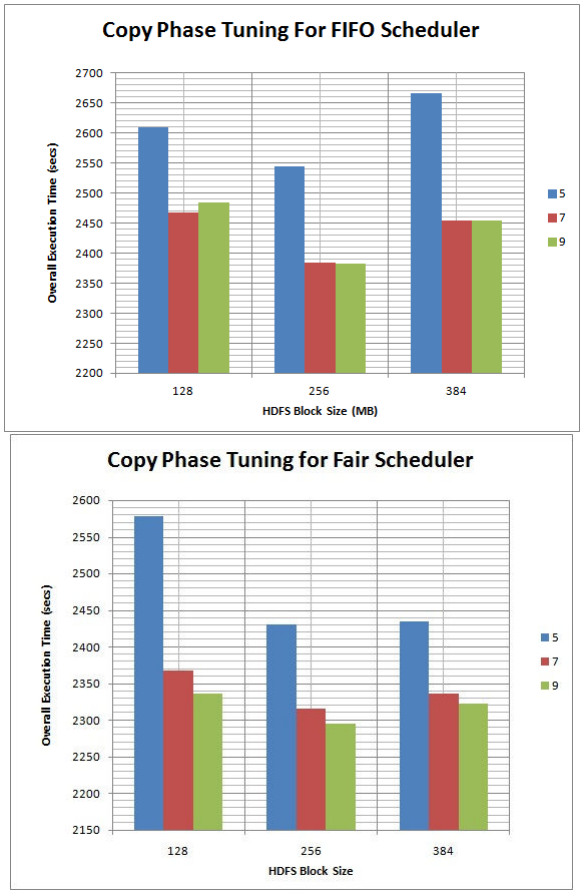

Fig. 3. Effect of tuning the copy phase on overall execution time

C. *Effect of Map Spill*

When the map function starts producing output, it is not simply written to disk. Each map task has a circular memory buffer that it writes the output to. The buffer is 100 MB by default, a size which can be tuned by changing the io.sort.mb property. When the contents of the buffer reaches a certain threshold size

(io.sort.spill.percent, default 0.80, or 80 %), a background thread will start to spill the contents to disk. If there is a limitation on available heap then one should try to minimize the number of spills by tuning io.sort.record.percent parameter values.

If the number of spilled records is greater than Map output records then additional spilling is occurring. An approach that can be taken to completely utilize the Map output buffer is to determine total size of the Map output and the total number of records contained in this output which can then be used to compute the space required for record buffer. The io.sort.mb property can then be configured to accommodate both the data and record buffer requirements. Fig. 4 shows the effects of the spill memory limit configuration. It can be seen that as the spill memory increases from certain threshold value, the execution time increases.

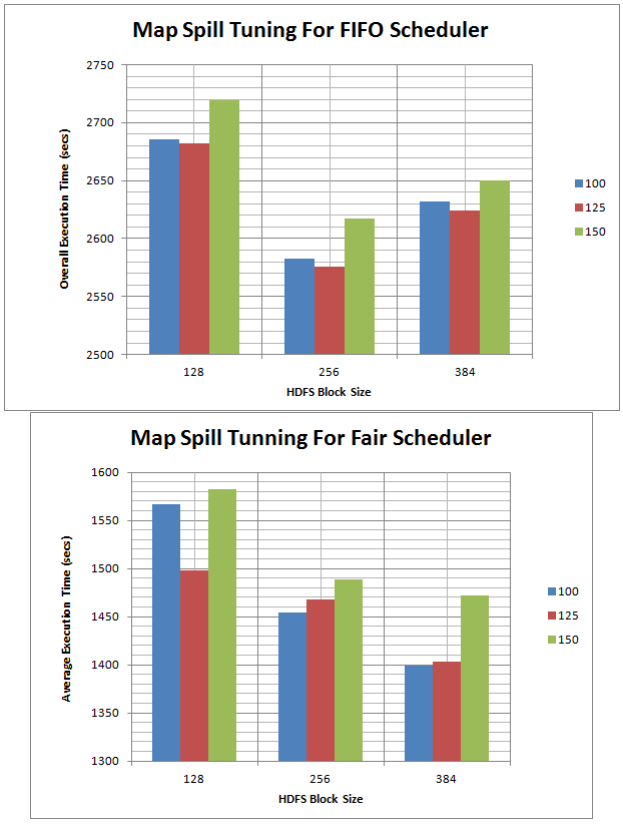

Fig. 4. Effect of Map spill tuning on overall execution time

D. Effect of Reduce Phase

Once Map tasks start completing their work, Map output gets sorted and partitioned per Reducer and is written to the disks of the TaskTracker node. These Map partitions are then copied over to the approximate Reducer Task Tracker. A buffer governed by mapred.job.shuffle.input.buffer.percent configuration parameter of mapred-site.xml, if big enough, will hold this Map output data. Otherwise, the Map output is spilled to the disks. The mapred.job.shuffle.input.buffer.percent is set to 0.70 by default. This means that 70 % of the Reduce JVM heap space will be reserved for storing copied Map output data. When this buffer reaches a certain threshold (governed by mapred.job.shuffle.merge.percent property) of occupancy the accumulated Map outputs are merged and spilled to the disk. Fig. 5 shows the effects of the Reduce JVM heap size on overall Execution Time. It can be seen that as we go on increasing the JVM heap the execution time goes on decreasing but after reaching certain threshold value it has no effect on overall execution time. But the execute time is lesser using 256 MB compared to 128 MB and 384 MB. Thus we observe that JVM heap size is also one of the main factor that effect the performance metrics.

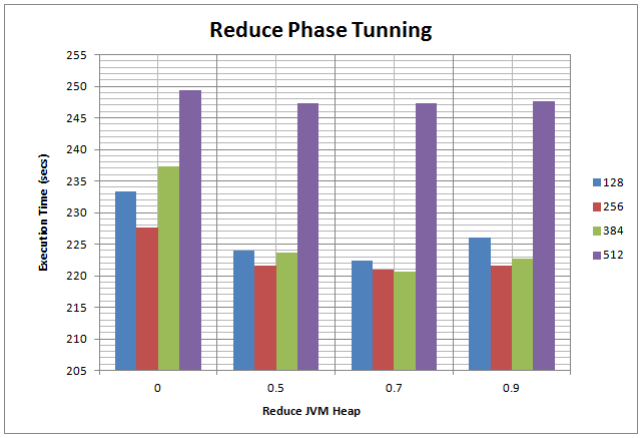

Fig. 5. Effect of reduce phase on overall execution time

IV. Framework for Fine Tuning Hadoop Configuration Parameters

One of the toughest problem in scheduling is understanding the factors contributing to the time taken to complete the job. Almost all the DAG scheduling algorithm heuristically predicts the time consumed and uses them to schedule the jobs or studying the time taken by a job from history of job executions logs. Using heuristics have been successful when the workloads are uniform and the underlying computing platform is homogeneous. Our contribution through this work is to understand the execution time of the task that belongs to

a job we progress in time. We in-turn use this time to understand the executions required for other tasks of a Job. Initially we schedule at least one tasks of a job to understand the computing time and the data transfer time required for the task to complete. By scheduling the first task, we can set the configuration parameters that best suits the job based on the history. We use this time to estimate the time required to complete other tasks of the same job. It should be noted that in Hadoop a Job is split into several tasks and each of which is scheduled and monitored through a scheduler. Thus our system dynamically learns the execution time of tasks in a job and accordingly schedule the tasks/jobs to achieve the goals.

Measuring the computation time in the non-homogeneous environment where every computing resources is different is a non-trivial tasks. We provide an approach by which we study the performance parameters as properties of these resources. The **value** of these properties are derived from the actual type of computing resources. In Map/Reduce framework, a Job is split into several tasks based on the number of data splits. These machine independent parameters are collected for only one task (which is scheduled first) for a job.

The values are extrapolated and computed for the individual computing resources on which the data split is placed. Thus the scheduler can compute the execution time for the other splits of data. These times are used for scheduling the job. Fig. 6 shows the details being collected from the task tracker (at the end of the completion of the tasks). As the actual execution of task is monitored by the Task tracker in the Map/Reduce framework, the task tracker collects these details for the tasks. The task tracker will have constant over head (time taken by executing commands) to collect the information and therefore is deterministic. In map/reduce framework, the task tracker reports the completion of the task to the job tracker. During the phase of reporting, the task tracker along with the completion time also reports the parameters and values collected for the specific tasks. This reporting the parameters collected for the tasks does not introduce any communication overhead. Further the collection of information is required to be performed only the first task that is being schedule.

Once the Job Tracker receives the parameters for the specific tasks, it then associates the task details to the Job details and passes this information to the scheduler. Along with the job details, it also sends the

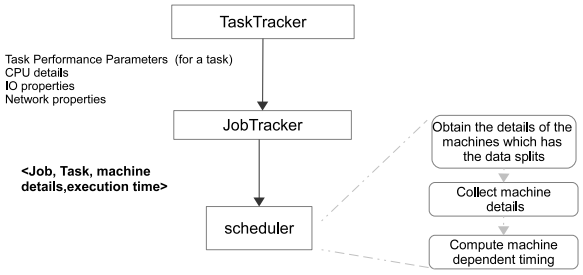

Fig. 6. System components reflecting the data exchanged and the computations performed by the Job Tracker

information about the machines on which the data splits are available for the jobs to be scheduled. It further computes the requirements of the tasks on these machines using the machine dependent parameters. The Job tracker provides the information about the network topology (for computing transfer times), task details, tasks resource requirements on machines in which the data splits are available to the scheduler. The scheduler consumes this information and schedules them such that throughput can be maximized and overall execution for the jobs can be minimized.

V. Related Work

There are several approached for performance tuning in Hadoop [2] addresses problem faced in *Terasort* benchmark. Paper present tuning methodologies and recommendations for varying workload conditions but the tuning recommendations made in the paper are based on optimizing the Hadoop *TeraSort* workload. In [3], authors explain tuning of Hadoop Parameters which directly affect Map-Reduce job performance under various conditions to achieve maximum performance. Similar approach under certain constraints is discussed in [4] where the solution stack maximizes productivity while limiting energy consumption and total cost of ownership. It also introduces some configuration and tuning advice that can help improve results in Hadoop environments. In [5], performance issues in heterogeneous Hadoop clusters are described.

VI. Conclusions and Future Work

We have studied extensively the parameters that affects the performance of Jobs in Hadoop clusters. We observe that several parameters along with different scheduling mechanisms have impact on the performance of the metrics. Our belief is that these parameters have to be efficiently measured for individual tasks and schedules

should be generated for maximizing the performance. We are in the process of building our framework where the scheduler will study the parameters for each tasks and incrementally use them to generate valid schedules. Our future work includes, parameter collection modules, generating valid schedules for work-flows (which are typically directed acyclic graphs).

REFERENCES

[1] Faraz Ahmad, Seyong Lee, Mithuna Thottethodi and T. N. Vijaykumar: PUMA: Purdue MapReduce Benchmarks Suite, Technical report, http://docs.lib.purdue.edu/ecetr/437/

[2] Dominique Heger, Hadoop Performance Tuning - A Pragmatic & Iterative Approach, Technical note: DH Technologies,2013, http://www.cmg.org/wp-content/uploads/2013/04/m_97_3.pdf.

[3] Shrinivas Joshi, Hadoop Performance Tuning Guide, AMD White Paper, 2012, http://developer.amd.com/wordpress/media/2012/10/Hadoop_Tuning_Guide-Version5.pdf.

[4] Intel, Optimizing Hadoop Deployments, Intel White Paper, 2010, http://www.intel.in/content/dam/doc/white-paper/cloud-computing-optimizing-hadoop-deployments-paper.pdf

[5] B.Thirumala Rao, N.V.Sridevi, V.Krishna Reddy and L.S.S.Reddy, Performance Issues of Heterogeneous Hadoop Clusters in Cloud Computing, Global Journal of Computer Science and Technology, Volume XI Issue VIII, May 2011

[6] Michael C. Schatz: CloudBurst, Bioinformatics Volume 25 Issue 11, June 2009.

[7] Slashdot report: http://slashdot.org/topic/bi/ibm-focuses-its-data-analytics-tools-on-employee-retention/

[8] Cloudera report: http://blog.cloudera.com/blog/2010/03/why-europes-largest-ad-targeting-platform-uses-hadoop/