AMD

Hadoop Performance Tuning Guide

Advanced Micro Devices

Table of Contents

REVISION HISTORY

Version	Date	Author	Changes
1.0	October 2012	Shrinivas Joshi	Initial draft

1.0 INTRODUCTION

Hadoop [1] is a Java based distributed computing framework that is designed to work with applications implemented using MapReduce programming model. Hadoop-MapReduce ecosystem software market is predicted to grow at 60.2% compound annual growth rate between 2011 and 2016 [2]. From small sized clusters to clusters with well over thousands of nodes, Hadoop technology is being used to perform myriad of functions – search optimizations, data mining, click stream analytics and machine learning to name a few. Unlike deploying Hadoop clusters and implementing Hadoop applications, tuning Hadoop clusters for performance is not a well-documented and widely-understood area. In this tuning guide, we attempt to provide the audience with a holistic approach of Hadoop performance tuning methodologies and best practices. Using these methodologies we have been able to achieve as much as 5.6X performance improvements. We discuss hardware as well as software tuning techniques including OS, JVM and Hadoop configuration parameters tuning.

Section 1 of this guide provides Introduction, challenges involved in tuning Hadoop clusters, performance monitoring and profiling tools that can aid in diagnosing bottlenecks, and the Hadoop cluster configuration that we used for the purpose of this study.

Section 2 contains a discussion on the steps that need to be taken to ensure optimal out-of-the-box performance of Hadoop clusters. Specifically, it talks about ways to ensure correctness of hardware setup, importance of upgrading Software components, stress testing benchmarks that can help uncover corner cases as well as ensure stability of the cluster and OS and Hadoop configuration parameters that could potentially impact successful completion of the Hadoop jobs.

Section 3 gives details of how the different Hadoop, JVM, and OS configuration parameters work, their relevance to Hadoop performance, guidelines on optimally tuning these configuration parameters as well as empirical data on the effect of these configuration tunings on performance of TeraSort workload while executed on the experiment setup used for this study.

Section 4 contains conclusions. Section 5 contains links to other helpful resources and Section 6 has References used in this document.

1.1 INTENDED AUDIENCE

This tuning guide is intended for Hadoop application developers interested in maximizing performance of their applications through configuration tuning at different levels of the Hadoop stack. System Administration professionals who are interested in tuning Hadoop cluster infrastructure will also benefit from the contents of this tuning guide.

This guide is intended for users who have intermediate level knowledge of the Hadoop framework and are familiar with associated technical terminology.

1.2 CHALLENGES INVOLVED IN TUNING HADOOP

Why is it difficult to optimally tune a Hadoop cluster?

Hadoop is a large and complex Software framework involving a number of components interacting with each other across multiple hardware systems. Bottlenecks in a subset of the hardware systems within the cluster can cause overall poor performance of the underlying Hadoop workload. Performance of Hadoop workloads is sensitive to every component of the stack - Hadoop, JVM, OS, network infrastructure, the underlying hardware, and possibly the BIOS settings. Hadoop has a large set of configuration parameters and a good number of these parameters can potentially have an impact on performance. Optimally tuning these configuration parameters requires a certain degree of knowledge of internal working of the Hadoop framework. Some parameters can have an effect on other parameter values. Thus one needs to adopt an iterative process of tuning where, implications of a particular configuration parameter on other parameters needs to be understood and taken into account before moving forward with further tuning. As with any large distributed system, identifying and diagnosing performance issues is an involved process. Overall, even though the Hadoop community makes every attempt to take the burden of tuning away from the users there are still a number of opportunities where user-driven tuning can help improve performance and in turn improve power consumption and total cost of ownership.

1.3 MONITORING AND PROFILING TOOLS

What are some of the tools that can help monitor and study performance bottlenecks of Hadoop jobs?

- Ganglia and Nagios: Ganglia [3] and Nagios [4] are distributed monitoring systems that can capture and report various system performance statistics such as CPU utilization, network utilization, memory usage, load on the cluster etc. These tools are also effective in monitoring overall health of the cluster.
- Hadoop framework logs: Hadoop task and job logs capture a lot of useful counters and other debug information that can help understand and diagnose job level performance bottlenecks.
- Linux OS utilities such as `dstat`, `vmstat`, `iostat`, `netstat` and `free` can help in capturing system-level performance statistics. This data can be used to study how different resources of the cluster are being utilized by the Hadoop jobs and which resources are under contention.
- Hadoop Vaidya [5] which is part of Apache Hadoop offers guidelines on mitigating Hadoop framework-level bottlenecks by parsing Hadoop logs.
- Java Profilers: Hprof profiling support that is built in to Hadoop, AMD CodeAnalyst [6] and Oracle Solaris Studio: Performance Analyzer [7] profiling tools can help in capturing and analyzing Java hot spots at Hadoop workload as well as Hadoop framework level.
- System level profilers –Linux perf [8] and OProfile [9] tools can be used for in-depth analysis of performance bottlenecks induced by hardware events.

1.4 METHODOLOGY AND EXPERIMENT SETUP

The tuning methodologies and recommendations presented in this guide are based on our experience with setting up and tuning Hadoop workloads on multiple clusters with varying number of nodes and configurations. We have experimented with hardware systems based on 3 different generations of AMD Opteron^{TM} processors. Some of these recommendations are also available in existing literature [10] [11] [12]. This guide however attempts to provide a systemic approach for tuning Hadoop clusters which encompasses all components of the Hadoop stack.

The empirical data presented in this guide is based on TeraSort [13] workload sorting 1TB of data generated using TeraGen workload. Needless to say that the nature of the Hadoop workloads will influence the degree to which some of these recommendations would show benefits. The tradeoffs of tuning recommendations mentioned in this guide should hopefully help the readers decide merit of particular recommendation in the workloads that they are interested in.

The Hadoop cluster configuration that we used for this study is as follows:

- 4 worker nodes (DataNodes and TaskTrackers), 1 master node (NameNode, SecondaryNameNode and JobTracker): 2 chips/16 cores per chip AMD Opteron^{TM} 6386 SE 2.8GHz
- 16 x 8 GB DDR3 1600 MHz ECC RAM per node
- 8 x Toshiba MK2002TSKB 2TB @7200 rpm SATA drives per node
- 1 x LSI MegaRAID SAS 9265-8i RAID controller per node
- 1 x 1GbE network interface card per node
- Red Hat Enterprise Linux Server release 6.3 (Santiago) with 2.6.32-279.5.2.el6.x86_64 kernel
- Oracle Java(TM) SE Runtime Environment (build 1.7.0_05-b06) with Java HotSpot(TM) 64-Bit Server VM (build 23.1-b03, mixed mode)
- Cloudera Distribution for Hadoop (CDH) 4.0.1 (version 2.0.0-mr1-cdh4.0.1) with custom hadoop-core*.jar which only adds the bug fix for https://issues.apache.org/jira/browse/MAPREDUCE-2374 bug.

2.0 GETTING STARTED

The business problem that needs to be solved fits the criteria of a Hadoop use-case, performance oriented development practices [10] [11] have been followed while implementing Hadoop jobs, initial capacity planning has been done to determine hardware requirements and finally the Hadoop jobs are up and running on a cluster that was just procured and deployed. How does one go about making sure that there are no Hardware and/or Software setup related bottlenecks that could impact out-of-the-box performance of the Hadoop cluster?

Based on our experience, following are some of the sanity and stability related steps the users could follow before deep diving into configuration tuning of Hadoop clusters:

- Verify that the hardware aspect of the cluster is configured correctly.
- Upgrade software components of the stack.
- Perform burn-in/stress tests.
- Tweak OS and Hadoop configuration parameters that impact successful completion of the Hadoop jobs.

Note that the tuning discussion in this guide is based on the Hadoop stack configuration mentioned in Section 1.4. Names and default values of some of the configuration parameters may vary with other versions of the software components. It is also likely that some of these parameters could be deprecated in newer versions.

2.1 CORRECTNESS OF HARDWARE SETUP

Apart from the configuration of the hardware systems, the BIOS, firmware and drivers used by the different components of the hardware system, the way hardware resources are viewed by the OS and the way certain components such as the memory DIMMs are installed on the system motherboard, amongst other things, can have noticeable impact on performance of the hardware systems.

To verify the correctness of the hardware setup we encourage customers to follow the guidelines as provided by the manufacturers of the different components of the hardware systems. This is a very important first step the users need to follow for configuring an optimized Hadoop cluster.

The default settings for certain performance impacting architectural features of the microprocessors and the platform as a whole are controlled by the BIOS. Bug fixes and improvements in out-of-the-box BIOS settings are made available via new BIOS releases. It is thus recommended to upgrade the system BIOS to the latest stable version.

System log messages should be monitored during cluster burn-in stages to rule out hardware issues with any of the components such as RAID/disk controller, hard disks, network controllers, memory DIMMs etc. We will talk about burn-in tests in a Section 2.3. Upgrading to the latest firmware and drivers for these components can help address performance as well as stability issues. Optimal IO subsystem performance is crucial for the performance of Hadoop workloads. Faulty hard drives identified in this process need to be replaced.

System memory configuration is also important in ensuring optimal performance and stability of the systems. The "AMD Opteron^{TM} 6200 Series Processors Linux® Tuning Guide" [14] provides a good set of guidelines for ensuring optimal memory configuration. Specifically, making sure that all the available memory channel bandwidth is utilized by the memory DIMMs, using optimal DIMM sizes and speed, verifying that the NUMA configuration view of the OS is correct and confirming baseline performance of the memory subsystem using benchmarks such as STREAM [15] are good measures for optimally configuring the memory subsystem. Instructions for downloading STREAM benchmark and correctly testing memory subsystem using this benchmark can also be found in "AMD Opteron^{TM} 6200 Series Processors Linux® Tuning Guide" document [14].

To summarize, the following steps need to be followed to correctly set up Hadoop cluster hardware infrastructure:

- Follow manufacturer's guidelines on optimally installing and configuring hardware components of the cluster.
- Upgrade to the latest, stable BIOS, firmware and drivers of the various components of the hardware systems.
- Perform benchmark tests to verify baseline performance of memory subsystem.

2.2 UPGRADING SOFTWARE COMPONENTS

The Linux® OS distribution version, the version of the Linux kernel bundled with the distribution, the Hadoop distribution version, the JDK version and the version of any other third-party libraries that the Hadoop framework and/or the Hadoop job depends on can impact out-of-the-box performance of the Hadoop cluster. Thus latest and stable software components of the Hadoop stack that allow correct functioning of the Hadoop jobs are recommended.

Performance, stability enhancements and bug fixes that get released with newer versions of Linux distributions can be important in improving performance of Hadoop. Default values of performance impacting kernel parameters are continuously evaluated by OS vendors to ensure that they are optimal for a broad range of software applications. The Linux kernel functionality is improved on a continuous basis and components such as the file systems and networking which play an important role in Hadoop performance are improved in this process.

The Hadoop ecosystem is evolving continuously. A number of bug fixes and performance enhancements get integrated in to the trunk code which then make into a stable release. In some cases, lack of performance features that leverage newer ISAs can cause significant regressions on some of the later microprocessors based systems. We recommend upgrading to the latest stable versions of Hadoop and the JVM that allow correct functioning of the underlying Hadoop workloads.

On newer generation hardware, Linux libraries such as glibc, libm and floating-point computation intensive libraries, among others, may perform poorly or could perform better if they are compiled to leverage newer ISA features. Also, certain libraries that get shipped with the Linux distributions may not be the best performing ones. See the blog entry [16]on how just a simple upgrade of the LZO [17] compression/decompression library can yield considerable performance improvements.

To summarize,

- Use the latest stable Linux distribution of your choice that is shown to improve performance without compromising stability and/or correct functioning of the underlying Hadoop workloads.
- Use the latest stable Hadoop distribution that helps improve performance of the underlying Hadoop workloads.
- Use the latest stable JVM and other third-party libraries that the underlying Hadoop workload depends on.

2.3 PERFORMING STRESS TESTS

Stress testing different sub-systems of a Hadoop cluster before moving Hadoop jobs to the production environment can help uncover corner cases of performance bottlenecks. These tests/benchmarks can also be used to measure and verify the baseline performance of different subsystems of the cluster. We recommend using both Hadoop as well as non-Hadoop benchmarks for these tests. While running these benchmarks for an extended period of time system logs can be monitored to check if there are indications of any cluster level performance bottlenecks. Following is a list of some of the benchmarks that can help in performing these tests:

- STREAM benchmark can be used to verify performance of memory subsystem and NUMA behavior of the hardware.
- IOzone file system benchmark can help measure baseline performance of the IO subsystem.

- Netperf can be used to stress test the network infrastructure and measure its baseline performance.
- Hadoop micro-benchmarks such as DFSIO, NNBench and MRBench can be used to stress test the setup of Hadoop framework. These micro-benchmarks are part of Hadoop distributions.
- Other benchmarks such as SPECcpu, SPECjbb® and SPECjvm® can also help in verifying the baseline performance of the systems.

2.4 ENSURING HADOOP JOB COMPLETION

Depending on the nature of the Hadoop workload default values of certain OS parameters and Hadoop configuration parameters could either cause Map/Reduce task and/or Hadoop job failures or could cause sizeable regression in performance. In our experience with TeraSort workload, following configuration parameters were of interest:

2.4.1 OS PARAMETERS

- The default maximum number of open file descriptors (FD) as configured using `ulimit` command can sometimes cause FDs to exhaust depending on the nature of the Hadoop workload. This causes exceptions that lead to job failures. We set this value to 32768 on our test cluster for this study.
- It is possible that fetch failures could occur while running Hadoop jobs with out-of-the-box settings. Sometimes these failures could be caused due to a lower value of `net.core.somaxconn` Linux kernel parameter. The default value of this parameter is 128. If required, this value needs to be increased to a sufficiently large value. We set this to 1024 on our test cluster.

2.4.2 HADOOP PARAMETERS

- Depending on the nature of the Hadoop workloads Map/Reduce tasks may not indicate their progress for a period of time which exceeds `mapred.task.timeout` Hadoop configuration property setting in mapred-site.xml. This property is set to 600 seconds by default. If required, the value of this property should be increased per the workload requirements. Note however that if there are genuine cases of task hang-ups due to hardware, cluster setup and/or workload implementation issues then increasing this value to an abnormally large value could mask it from Hadoop framework. This in turn can cause performance regressions.
- If `java.net.SocketTimeoutException` exceptions which say something like “timeout while waiting for channel to be ready for read/write” are encountered then `dfs.socket.timeout` and `dfs.datanode.socket.write.timeout` property values in hdfs-site.xml need to be increase to an acceptable level. Again the tradeoff here is that a genuine case of timeout which is independent of the value of these properties may go unnoticed for more than acceptable period of time.

3.0 PERFORMANCE TUNING

Once hardware and software components of the Hadoop cluster are verified to be functioning at the best possible out-of-the-box performance level one can start deep diving in to fine tuning of configuration knobs. Configuration parameters at all the levels of the Hadoop stack – Hadoop framework, JVM, and OS can significantly impact performance of the Hadoop workloads. In this section, based on our experience with tuning TeraSort workload, we will provide a set of guidelines that can help tune and maximize performance of Hadoop workloads. The steps in these guidelines can be followed in the sequence that they are presented to achieve maximum performance benefits. Wherever appropriate we also provide empirical data that shows the performance impact of the recommendations made in these guidelines. The following sections will cover guidelines for tuning configuration parameters for the Hadoop framework, JVM, and OS. Note that the Hadoop tuning recommendations in this guide are applicable to the Hadoop distribution and version as mentioned in Section 1.3.

3.1 HADOOP CONFIGURATION TUNING

In this section we will talk about how to come up with a baseline configuration for the Hadoop workload and then how to move forward with tuning this configuration to achieve maximum levels of resource utilization and performance.

As mentioned earlier we will be using TeraSort as our example workload here. In the absence of compression, the Map phase of TeraSort workload does 1TB worth of disk IO read and 1TB worth of disk IO write operations, excluding the IO operations that result from spills and speculative execution. The Reduce phase also does 1TB worth of disk IO read and 1TB worth of disk IO write operations, excluding the IO operations resulting from Reduce-side spills and speculative execution. Overall, TeraSort spends a big chunk of its execution time performing IO operations. Depending on the nature of the Hadoop workload some of the guidelines presented in following sections may or may not show optimal results.

3.1.1 BASELINE CONFIGURATION

The default number of Map/Reduce slots and the Java heap size options may not be sufficient for some Hadoop workloads. Thus the first step in defining a baseline configuration is to modify these configuration parameters to an acceptable level. The goal here is to maximize the CPU and other hardware resources utilization of the cluster. The Java heap size requirements of the Hadoop workload, the number of Map/Reduce tasks spawned by the workload, the number of hardware cores/threads available on the systems, the available IO bandwidth and the amount of available memory will all influence the baseline configuration. A methodology that has helped us in our experience is to start with:

A. Sufficient number of hard disk drives that satisfies the anticipated storage requirements of the underlying workload
B. Configure the number of Map and Reduce slots such that Map phase gets as much CPU resources as possible and all the processor cores are utilized during the Reduce phase as well
C. Configure the Java heap sizes for Map and Reduce JVM processes such that enough memory is left for OS buffers and caches after fulfilling Map/Reduce task's heap requirements.

The relevant Hadoop configuration parameters here are `mapred.map.tasks`, `mapred.tasktracker.map.tasks.maximum`, `mapred.reduce.tasks`, `mapred.tasktracker.reduce.tasks.maximum`, `mapred.map.child.java.opts`, and `mapred.reduce.child.java.opts` and can be found in mapred-site.xml file. Using this approach, for the baseline configuration, we decided to:

A. Use 4 data disk drives per DataNode

B. Allocate 2 Map slots and 1 Reduce slot per hardware core
C. Allocate 1GB of initial and max Java heap size for Map and Reduce JVM processes.

Thus we spawn 3 JVM processes per hardware core at any given time consuming as much as 3GB heap per hardware core. We had 4GB of RAM per hardware core available on the systems. The rest 1GB of memory per hardware core was left for OS buffers and caches. Using this configuration as baseline configuration we started to look at the tuning opportunities. Note that the baseline configuration was derived with some knowledge of working of TeraSort workload. Baseline configuration is expected to differ depending on characteristics of the underlying Hadoop workload.

3.1.2 DATA DISK SCALING

The next step after defining the baseline configuration should be to understand how well the workload scales with the number of available data disks. One needs to configure `mapred.local.dir` in mapred-site.xml and `dfs.name.dir` and `dfs.data.dir` in hdfs-site.xml to change the number of data disks used by the Hadoop framework. Given the IO heavy nature of TeraSort workload we saw the performance scaling well with increased number of disks. Figure 1 contains a graph that shows the normalized performance difference of TeraSort for different number of data disks.

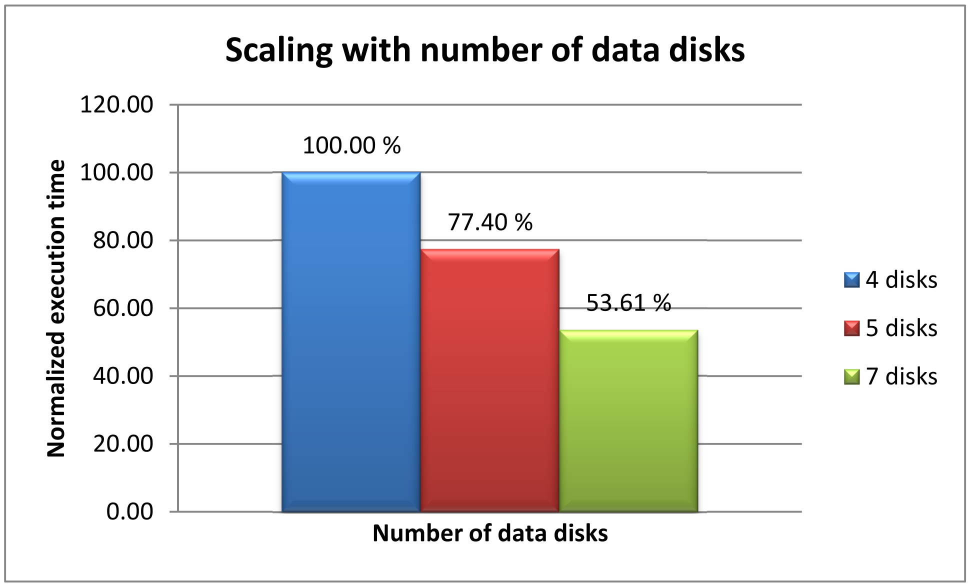

Figure 1: TeraSort performance scaling with number of data disks

3.1.3 COMPRESSION

Hadoop supports compression at 3 different levels - input data, intermediate Map output data and Reduce output data. It also supports multiple codecs that can be used for performing compression and decompression. Some codecs have a better compression factor but take longer to compress as well as to decompress. Some codec strike a fine balance between the compression factor and the overhead of compression and decompression activities. TeraSort workload does not support input data compression neither does it support Reduce output data compression. Therefore we could just experiment with compressing intermediate Map output. Enabling Map output compression reduces the disk and network IO overhead at the expense of CPU cycles necessary for compression and decompression. Thus, the tradeoffs of compression/decompression at all the different supported possible levels of the Hadoop workloads should be evaluated. The properties one should be interested in here are – `mapred.compress.map.output`, `mapred.map.output.compression.codec`, `mapred.output.compress`, `mapred.output.compression.type`, and `mapred.output.compression.codec` found in mapred-site.xml. The tradeoff here will only be in cases where CPU resources are limited and that the underlying workload is very CPU intensive where the processor doesn't stall for IO activities during which compression/decompression threads can receive cycles on the CPU.

The graph in Figure 2 shows the effect of Map output compression on TeraSort performance with different codec choices.

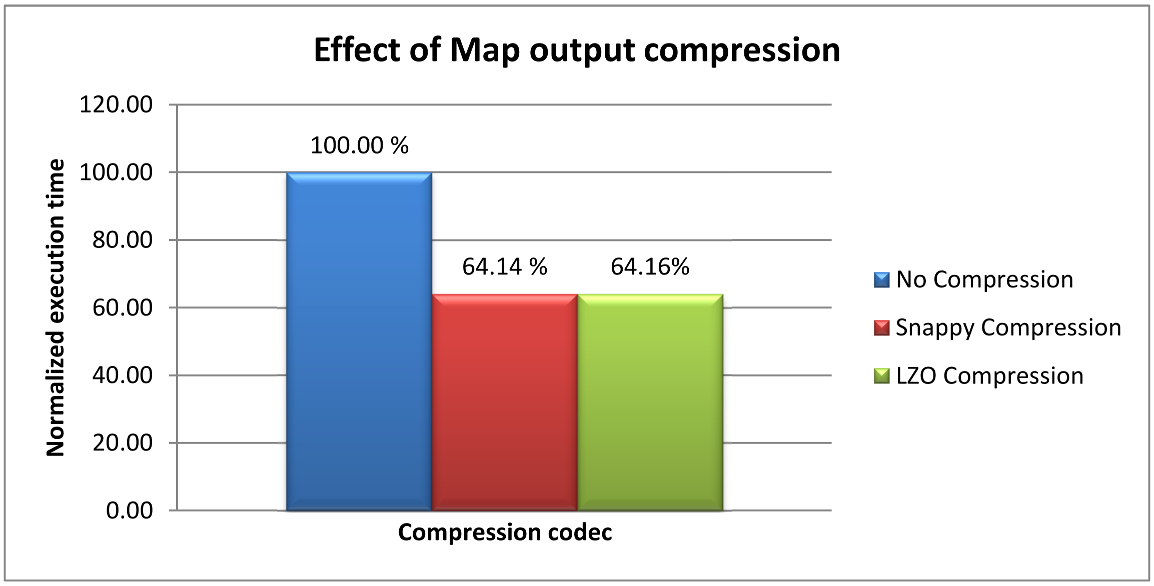

Figure 2: Effect of Map output compression using different codecs on TeraSort performance

We noticed about 28% run-to-run variance while using Snappy codec. The numbers for Snappy codec in Figure 2 were derived by excluding variance causing runs. Since we observed much smaller variance (1.5%) with LZO codec, we decided to continue using it as our choice of best performing codec.

3.1.4 JVM REUSE POLICY

Hadoop supports a configuration parameter called `mapred.job.reuse.jvm.num.tasks` which governs whether Map/Reduce JVM processes spawned once are re-used for running more than 1 task. This property can be found in mapred-site.xml and the default value of this parameter is 1 which means that the JVM is not re-used for running multiple tasks. Setting this value to -1 indicates that an unlimited number of tasks can be scheduled on a particular JVM instance. Enabling JVM reuse policy reduces the overhead of JVM startup and teardown. It also improves performance since the JVM spends less time interpreting Java bytecode since some of the earlier tasks are expected to trigger JIT compilation of hot methods. JVM reuse is expected to specifically benefit in scenarios where there are large number of very short running tasks. We noticed close to 2% improvement in performance by enabling JVM reuse.

3.1.5 HDFS BLOCK SIZE

Each Map task operates on what is called an input split. The `mapred.min.split.size` (of mapred-site.xml), `dfs.block.size` (of hdfs-site.xml) and `mapred.max.split.size` (of mapred-site.xml) configuration parameters decide the size of the input split. The input split size and the total input data size of the Hadoop workload determines the total number of Map tasks spawned by the Hadoop framework. For workloads like TeraSort the most natural way to change the input split size is by changing the HDFS block size value using `dfs.block.size` parameter. If the Hadoop job is spawning a large number of Map tasks experiment with larger HDFS block sizes. Reducing the number of Map tasks this way can decrease the overhead of starting up and tearing down of Map JVMs. It can also reduce the cost involved in merging map output segments during the Reduce phase. Larger block sizes also help increase the execution time taken by each Map task. It is better to run small number of longer running Map tasks rather than large number of very short running Map tasks. Note that if Map output size is proportional to the HDFS block size then bigger block size could lead to additional Map-side spills if spill related properties are not adjusted accordingly. Figure 3 shows how performance varies with different block sizes. We found 256M to be the optimal size for our configuration.

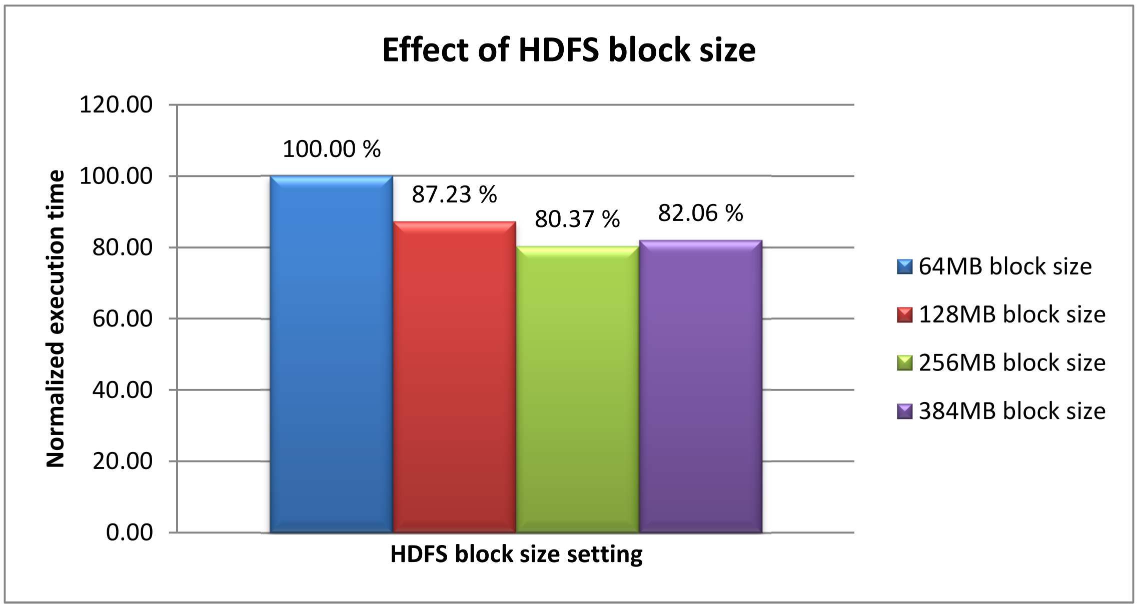

Figure 3: TeraSort performance comparison with different HDFS block sizes

3.1.6 MAP-SIDE SPILLS

While Map tasks are running the generated intermediate output is stored into a buffer. This buffer is a chunk of reserved memory that is part of Map JVM heap space. The default size of this buffer is 100 MB. This is governed by the value of `io.sort.mb` (of mapred-site.xml) configuration parameter. A part of this buffer is reserved for storing metadata for the spilled records. By default this is set to 0.05 (5%) of `io.sort.mb` and hence it equals to 5MB. This is governed by the `io.sort.record.percent` configuration parameter of mapred-site.xml. Each metadata record is 16 bytes long. Thus metadata for a total of 327680 records can be stored in the record buffer. Contents of this buffer are spilled to the disks as soon as a certain threshold of occupancy is reached by either the output data part of the buffer or the metadata part of the buffer. This threshold which is governed `io.sort.spill.percent` configuration parameter of mapred-site.xml is set to 0.8 (80%) by default.

Spilling Map output to the disk multiple times (before the final spill) can lead to additional overhead of reading and merging of the spilled records. The default values of the properties affecting Map-side spill behavior are only optimal for ~304 bytes long map input records. If there is sufficient Java heap memory available for Map JVMs, the goal here should be to eliminate all intermediate spills and just spill the final output. 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. An easy way to detect if the Map phase is performing additional spills is to look at “Map output records” and “Spilled Records” counter of JobTracker web interface for the job immediately after the Map phase is complete. 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 and the `io.sort.spill.percent` value can be set to something like 0.99 to let the buffer fill nearly to its capacity since we will be doing first and final spill once the Map task is complete. Note that the size of this buffer would need to be increased slightly over the total buffer space requirements. We observed spilling when we didn't allow additional space on top of Map output data and record buffer requirements. For example, for HDFS block size setting of 256MB where each record is 100 bytes long we set `io.sort.mb` to 316MB, `io.sort.record.percent` to 0.162 (16.2% of 316MB) and `io.sort.spill.percent` to 0.99 (99% fill threshold) to completely eliminate Map-side spills. We noticed 2.64% improvement in execution time from eliminating Map-side spills

3.1.7 COPY/SHUFFLE PHASE TUNING

If it is noticed that the Reduce phase doesn't complete copying map outputs soon after all the Map tasks are executed then this can be an indication of a poorly configured cluster. There could be multiple potential causes for a slow progressing copy phase:

- The maximum number of parallel map-output copier threads governed by `mapred.reduce.parallel.copies` in mapred-site.xml is set to 5 by default. This could be a limiting factor for the throughput of copy operation.
- The maximum number of worker threads at TaskTracker level that are used to serve map outputs to the reducers is governed by tasktracker.http.threads and is set to 40 by default. This is a TaskTracker level property. One could experiment increasing this value in steps to see its effect on copy phase.
- Configuration parameters such as `dfs.datanode.handler.count` (found in hdfs-site.xml), `dfs.namenode.handler.count` (found in hdfs-site.xml) and `mapred.job.tracker.handler.count` (found in mapred-site.xml) could also be explored for further fine-tuning.
- Reduce side bottlenecks could also slow down the copy phase progress. Tuning targeted at avoiding disk spills on the Reduce side could speed up the copy phase. In our case it did turn out to be Reduce side bottlenecks that were slowing down the copy phase.
- Network hardware related bottlenecks could also be a contributing factor. Using benchmarks like Netperf, one could confirm that the network bandwidth across all the worker nodes is at a practically acceptable level.

3.1.8 REDUCE-SIDE SPILLS

The Reduce phase can be crucial in influencing the total execution time of Hadoop workloads. Reduce phase is more network intensive and could possibly be more IO intensive depending on the nature of the Hadoop workload. After all, all the output generated by potentially large number of Map tasks needs to be copied, aggregated/merged, processed and possibly all this data needs to be written back to the HDFS. Thus depending on the total number of Reduce slots allocated to the Hadoop job, the Java heap requirements of Reduce JVMs can be much more demanding than the Map JVMs especially when optimal performance is a consideration.

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 appropriate Reducer TaskTrackers. 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 mapred-site.xml which is 0.66 by default) of occupancy the accumulated Map outputs are merged and spilled to the disk. Once the Reduce-side sort phase is over a part of Reduce JVM heap governed by `mapred.job.reduce.input.buffer.percent` of maped-site.xml can be used to retain Map outputs before feeding it into the final reduce function of the Reduce phase. The `mapred.job.reduce.input.buffer.percent` parameter is set to 0.0 by default which means that all the Reduce JVM heap is allotted to the final reduce function.

Essentially, having large mapred.job.shuffle.input.buffer.percent and `mapred.job.reduce.input.buffer.percent` buffers will aid in avoiding unnecessary IO operations caused by Reduce-side spills. In cases of identity Reducers such as the one used by TeraSort workload the reduce function is not expected to require large Java heap memory. In such scenarios performance improvements can be expected by increasing `mapred.job.reduce.input.buffer.percent` value to as close as possible to 1.0

We saw performance improvements by increasing mapred.job.shuffle.input.buffer.percent parameter to 0.9 and `mapred.job.reduce.input.buffer.percent` to 0.8. It was clear that the Reduce phase of TeraSort could thrive on additional heap. At this point we decided to evaluate the tradeoff between having 2 Map slots per hardware core vs. having only 1 Map slot per hardware core and allotting all the freed up Map Java heap space to the Reduce JVMs. Large heap space on Reduce side noticeably improved the performance.

We noticed ~10% improvement from Reduce phase tuning. Figure 4 shows this data in form of a graph.

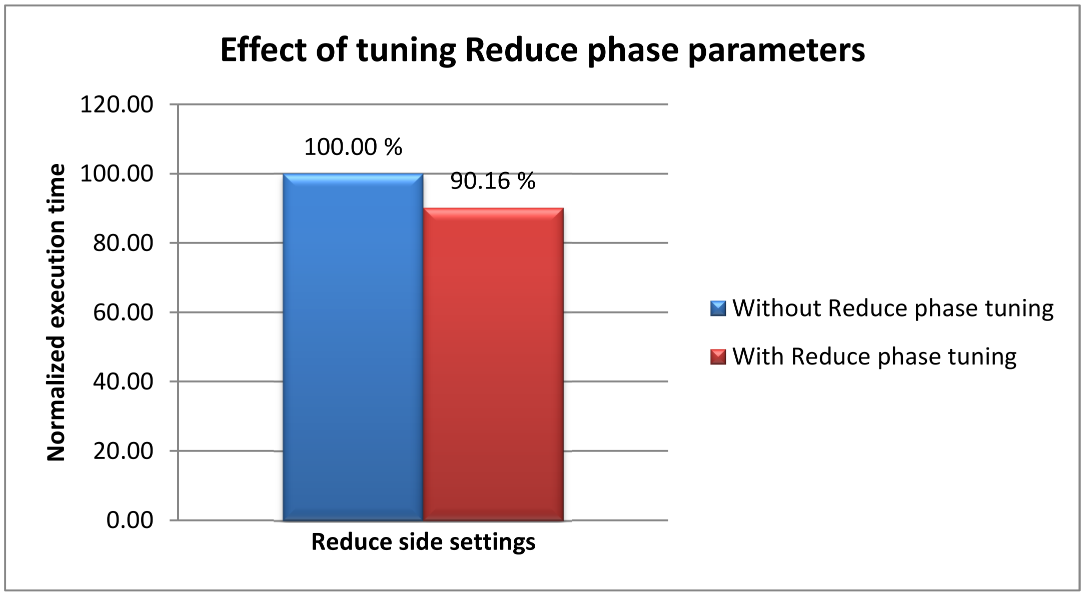

Figure 4: Effect of tuning Reduce phase Hadoop parameters on TeraSort

3.1.9 POTENTIAL LIMITATIONS

Finally, there could be certain known limitations or bugs in the Hadoop environment being used that could cause performance and/or stability issues. For example, CDH 4.0.1 and earlier releases are affected by a bug [18] in MR framework where Map/Reduce task failures can occur for a small subset of tasks. These failures can induce large regressions and variance in execution time. For example, if long running Reduce tasks fail due to this bug and if there are no additional Reduce slots available then the failed task will not be re-scheduled by the Hadoop framework until there is a free Reduce slot available. Also, the `mapred.max.tracker.failures` configuration parameter of mapred-site.xml which governs when a TaskTracker gets blacklisted is set to 4 by default. This means that if any of the TaskTrackers have 4 or more task failures then those TaskTrackers will be blacklisted and no further scheduling of tasks will happen on them. To get around these regressions one can set the value of `mapred.max.tracker.failures` property to a sufficiently large value or use a version of the Hadoop distribution that fixes this issue. Care needs to be taken to make sure that the `mapred.max.tracker.failures` property does compromises stability of the cluster. On our setup, we built

Hadoop core jar files from the source after including the fix for this issue. Note that the recently released CDH 4.1 does include the fix for this bug.

Thus it is important to identify if there are any known issues that could potentially harm performance and/or stability of the cluster.

3.2 JVM CONFIGURATION TUNING

Once all the possible bottlenecks on the Hadoop side of the stack have been addressed or at least mitigated to a great extent it is worthwhile looking at further tuning the JVM.

3.2.1 JVM FLAGS

JVM vendors strive to improve out-of-the-box performance of the JVMs so that the users do not have to perform flag mining experiments for their applications. There is however still the possibility of seeing improved performance via JVM flag tuning without spending extensive efforts in this exercise. Following are some of the flags worth exploring:

- AggressiveOpts – This is an umbrella flag that governs whether certain other optimizations are enabled by the JVM or not. The optimizations enabled by these flags could vary depending on the JVM version and hence it is important to test with this flag especially when the systems are upgraded to a newer JVMs. On our cluster we saw ~3.4% improvement by enabling aggressive optimizations feature. Note that AggressiveOpts is disabled by default on Oracle JDK7 Update 5.
- UseCompressedOops – Compressed Ordinary Object Pointers or simply compressed pointers help reduce memory footprint of 64-bit JVMs at the expense of reduced total addressable Java heap space. This feature is enabled by default in newer Oracle JVM versions. If an older version of the JVM is being used and compressed pointers are disabled, experiment with enabling them. We saw a less than 1% regression on TeraSort performance by disabling this feature. Note that UseCompressedOops is enabled by default on Oracle JDK 7 Update 5.
- UseBiasedLocking - Biased-locking feature in Oracle HotSpot JDK improves performance in situations where locks are generally un-contended. We saw a less than 1% regression on TeraSort performance by disabling this feature. Note that UseBiasedLocking is enabled by default on Oracle JDK 7 Update 5.

3.2.2 JVM GARBAGE COLLECTION

Garbage collection analysis and tuning of Map and Reduce JVMs would not be as straightforward as experimenting with different JVM flags. However, if one is fairly acquainted with GC analysis and tuning then the gains achieved by tuning GC behavior could offset the time and efforts involved in this exercise. On a test-bed different than the one used for the experiments in this tuning guide we have observed modest improvements in performance by tuning GC behavior of Map and Reduce JVMs. Readers interested in additional details of GC tuning are encouraged to read this article.

Figure 5 shows the performance improvements achieved via JVM tuning.

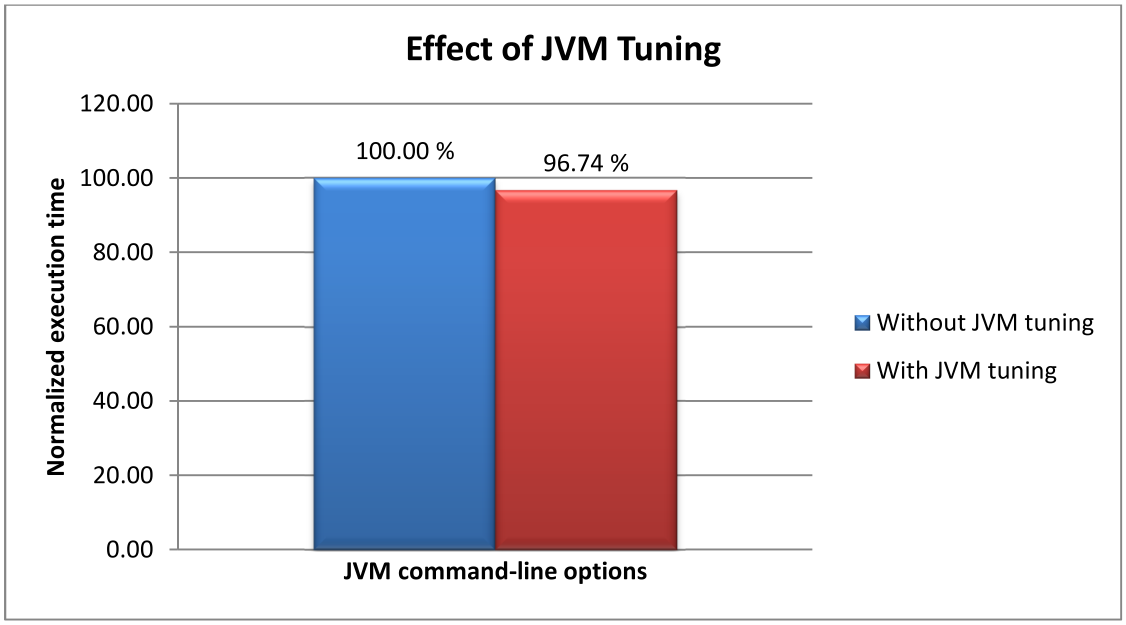

Figure 5: Effect of JVM command-line options tuning on TeraSort

3.3 OS CONFIGURATION TUNING

In this section we discuss OS configurations that could have positive impact on performance of Hadoop workloads.

3.3.1 TRANSPARENT HUGE PAGES

Transparent huge pages (THP) feature of RHEL 6.2 and above aims at simplifying management of huge pages. This feature has shown to improve out-of-the-box performance of a wide range of applications. However, Hadoop workloads while running with THP feature enabled have shown high kernel space CPU utilization caused by the compaction process of THP feature. This can cause large regressions. We have noticed as much as 66% regression while running TeraSort workload with THP enabled. THP feature needs to be disabled incase similar symptoms are observed while running the Hadoop workloads. See “Known Issues and Work Arounds in CDH4” section of CDH 4.0.1 release notes [19] for additional details. A tradeoff here is that if other production applications on the same cluster benefit from THP then their performance will get affected.

3.3.2 FILE SYSTEM CHOICE AND ATTRIBUTES

The default file system (FS) supported by Linux can vary depending on the distribution and the version of the distribution. Given the fact that internals of FS are analyzed and improved upon on a continuous basis, the choice of FS can have significant impact on performance of Hadoop workloads especially for IO intensive workloads. RHEL 6.3 supports EXT4 as the default FS and based on our past experience we have found it to be significantly better performing in comparison to EXT3 FS.

In the absence of noatime attribute, every file read operation triggers a disk write operation for maintaining last access time of the file. This metadata logging can be disabled by adding noatime (which also implies nodiratime) attribute to the FS mount options of the data disks. We noticed 29% improvement in performance of TeraSort by using noatime FS mount attribute.

3.3.3 IO SCHEDULER CHOICE

Majority of Linux distributions support 4 different types of IO schedulers – CFQ, deadline, noop and anticipatory. Depending on pattern of IO operations, the hardware configuration of IO subsystem and the requirements of the application a particular IO scheduler may perform better in comparison to the others. The default IO scheduler could vary depending on the Linux distribution. For example Ubuntu 11.04 has deadline IO scheduler as the default scheduler whereas RHEL 6.3 has CFQ as the default scheduler. As observed in our previous set of experiments in http://dl.acm.org/citation.cfm?id=2188323 we noticed as much as 15% improvement by using CFQ scheduler instead of deadline scheduler.

Chart in Figure 6 shows the total improvements from OS tuning

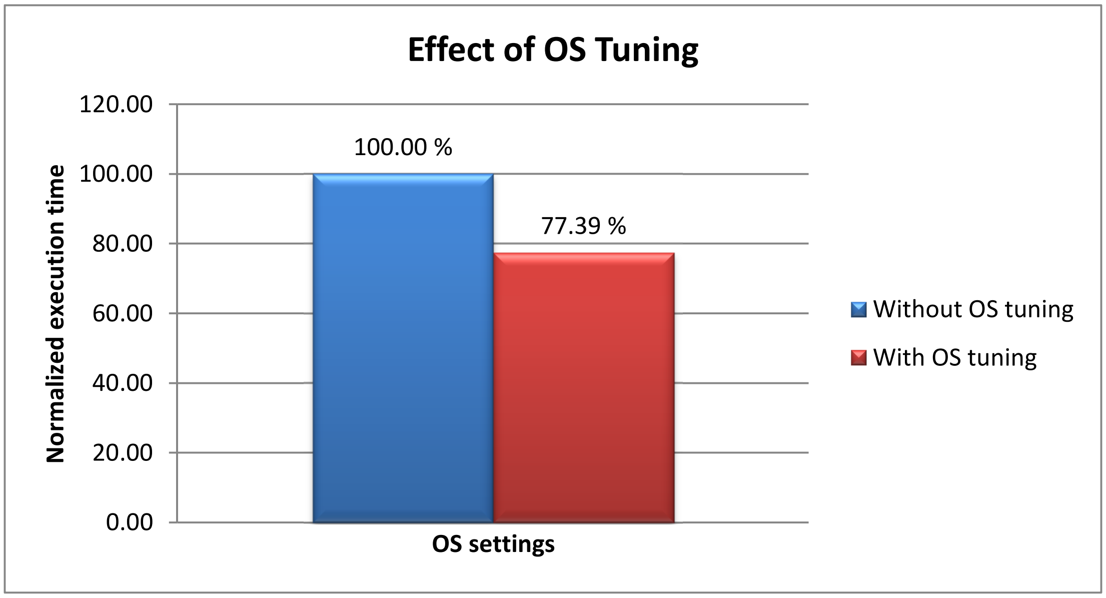

Figure 6: Effect of OS configuration tuning on TeraSort performance

4.0 FINAL WORDS AND CONCLUSION

BigData analytics and Hadoop ecosystem are increasingly catching attention of businesses and their IT organizations. The core Hadoop framework is the brain of the nervous system of this ecosystem. The open source community and commercial vendors of Hadoop are pushing innovations into this ecosystem at a rapid pace. The Hadoop technology is at a stage where businesses have realized the value proposition of this technology and have either already invested resources in it or are seriously evaluating the proposition. Users who are relatively new to the Hadoop technology would certainly welcome knowhow on proven methodologies to improve stability and performance of Hadoop clusters.

In this tuning guide we have demonstrated how a systematic approach can be adopted to optimally tune all the different components of the Hadoop stack. On TeraSort workload we were able to achieve as much as 5.6X gains when a minimal hardware configuration (4 disks configuration of Figure 1) is considered as the baseline configuration. We were also able to achieve a 3X improvement when the baseline configuration (7 disks configuration of Figure 1) uses all available resources of the hardware systems.

Figure 7 depicts the performance delta between baseline configurations and tuned configuration:

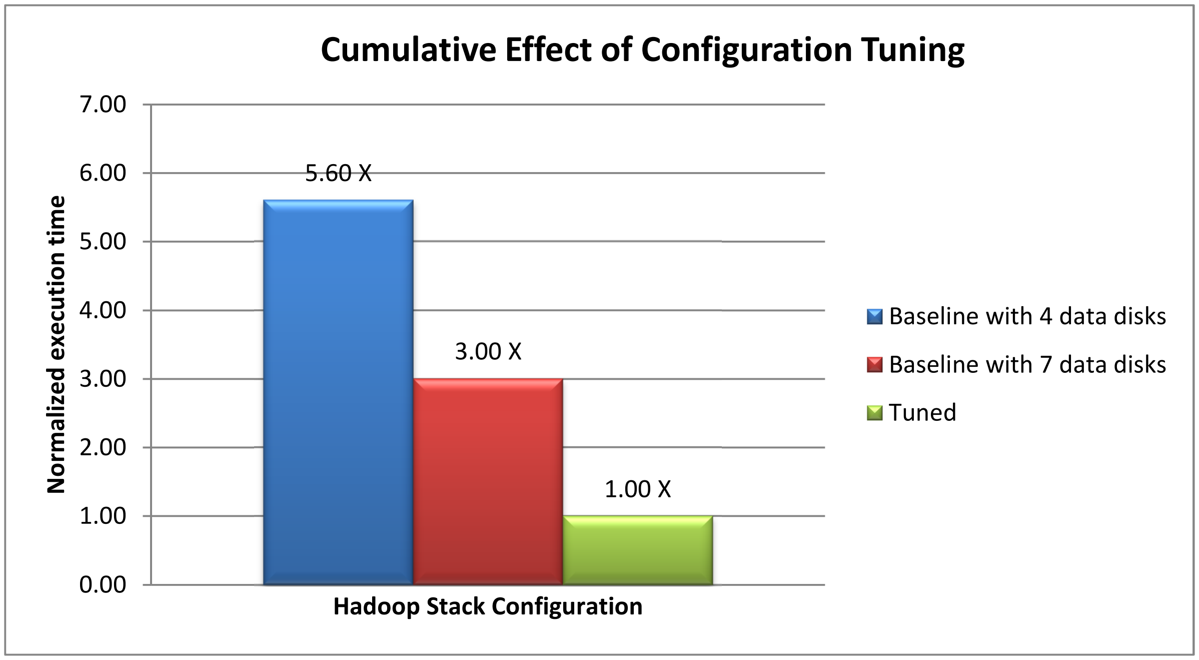

Figure 7: Total improvements in TeraSort performance through configuration tuning

Setting up Hadoop clusters and tuning them so that they are stable and well performing is an involved process. We hope that the recommendations in this guide make this process a little easier.

5.0 RESOURCES

- AMD Developer Central Java Zone: http://developer.amd.com/Resources/documentation/java/Pages/default.aspx
- AMD Developer Tools: http://developer.amd.com/tools
- AMD Opteron^{TM} 6200 Series Processors Linux® Tuning Guide: http://blogs.amd.com/developer/2012/04/25/oracle-java-and-allocateprefetchstyle-for-%E2%80%9Cbulldozer%E2%80%9D-processors/
- 5 minutes of work for up to 5% more Hadoop performance: http://blogs.amd.com/developer/2011/10/20/5-minutes-work-for-up-to-5-more-hadoop-performance/
- Java Garbage Collection Characteristics and Tuning Guidelines for Apache Hadoop TeraSort workload: http://developer.amd.com/Resources/documentation/java/Pages/default.aspx
- Apache hadoop performance-tuning methodologies and best practices: http://dl.acm.org/citation.cfm?id=2188323
- Apache Hadoop Performance-tuning Methodologies and Best Practices – a downloadable reference guide: http://blogs.amd.com/developer/2011/07/12/apache-hadoop-performance-tuning-methodologies-and-best-practices-%E2%80%93-a-downloadable-reference-guide/
- Mahout Recommendation Engine – Threshold flag: http://blogs.amd.com/developer/2012/06/06/mahout-recommendation-engine-%E2%80%93-threshold-flag/
- MapReduce Optimization in Mahout Recommendation Engine: http://blogs.amd.com/developer/2012/05/29/mapreduce-optimization-in-mahout-recommendation-engine/
- Oracle Java and AllocatePrefetchStyle for “Bulldozer” Processors: http://blogs.amd.com/developer/2012/04/25/oracle-java-and-allocateprefetchstyle-for-%E2%80%9Cbulldozer%E2%80%9D-processors/

6.0 REFERENCES

[1] "Welcome to Apache Hadoop," [Online]. Available: http://hadoop.apache.org/. [Accessed 07 October 2012].

[2] "IDC Releases First Worldwide Hadoop-MapReduce Ecosystem Software Forecast, Strong Growth Will Continue to Accelerate as Talent and Tools Develop - prUS23471212," [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS23471212. [Accessed 07 October 2012].

[3] "Gangia Monitoring System," [Online]. Available: http://ganglia.sourceforge.net/. [Accessed 07 October 2012].

[4] "Nagios - The Industry Standard in IT Infrastructure Monitoring," [Online]. Available: http://www.nagios.org/. [Accessed 07 October 2012].

[5] "Vaidya Guide," [Online]. Available: http://hadoop.apache.org/docs/mapreduce/current/vaidya.html. [Accessed 07 October 2012].

[6] "AMD CodeAnalyst Performance Analyzer | AMD Developer Central," [Online]. Available: http://developer.amd.com/tools/hc/CodeAnalyst/pages/default.aspx. [Accessed 07 October 2012].

[7] "Oracle Solaris Studio 12.2: Performance Analyzer - Oracle Solaris Studio 12.2: Performance Analyzer," [Online]. Available: http://docs.oracle.com/cd/E18659_01/html/821-1379/index.html. [Accessed 07 October 2012].

[8] "Main Page - Perf Wiki," [Online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page. [Accessed 07 October 2012].

[9] "About OProfile," [Online]. Available: http://oprofile.sourceforge.net/about/. [Accessed 07 October 2012].

[10] T. White, Hadoop: The Definitive Guide, Sebastopol: O'Reilly Media, Inc., 2010.

[11] "7 Tips for Improving MapReduce Performance | Apache Hadoop for the Enterprise | Cloudera," [Online]. Available: http://www.cloudera.com/blog/2009/12/7-tips-for-improving-mapreduce-performance/. [Accessed 07 October 2012].

[12] "Developer Blog: Setting up a Hadoop cluster - Part 1: Manual Installation," [Online]. Available: http://gbif.blogspot.com/2011/01/setting-up-hadoop-cluster-part-1-manual.html. [Accessed 07 October 2012].

[13] O. O'Malley. [Online]. Available: http://sortbenchmark.org/YahooHadoop.pdf. [Accessed 07 October 2012].

[14] "Developer Guides and Manuals | AMD Developer Central," [Online]. Available: http://developer.amd.com/Assets/51803A_OpteronLinuxTuningGuide_SCREEN.pdf. [Accessed 07 October 2012].

[15] "MEMORY BANDWIDTH: STREAM BENCHMARK PERFORMANCE RESULTS," [Online]. Available: https://www.cs.virginia.edu/stream/. [Accessed 09 October 2012].

[16] "5 minutes work for up to 5% more Hadoop performance | AMD Developer Central," [Online]. Available: http://blogs.amd.com/developer/2011/10/20/5-minutes-work-for-up-to-5-more-hadoop-performance/. [Accessed 09 October 2012].

[17] "oberhumer.com: LZO real-time data compression library," [Online]. Available: http://www.oberhumer.com/opensource/lzo/. [Accessed 09 October 2012].

[18] "[#MAPREDUCE-2374] "Text File Busy" errors launching MR tasks - ASF JIRA," [Online]. Available: https://issues.apache.org/jira/browse/MAPREDUCE-2374. [Accessed 09 October 2012].

[19] "CDH4 Release Notes - Cloudera Support," [Online]. Available: https://ccp.cloudera.com/display/CDH4DOC/CDH4+Release+Notes. [Accessed 09 October 2012].