# Intel® Distribution for Apache Hadoop* Software: Optimization and Tuning Guide

Configuring and managing your Hadoop* environment for performance and cost

## Table of Contents

Executive Summary . . . . . . . . . . . . . . 1
Introduction . . . . . . . . . . . . . . . . . . . . . 1
Components of Intel® Distribution for Apache Hadoop* Software . . . . . 2
Resource Recommendations . . . . . . 3
Optimizing and Tuning the Hadoop* System . . . . . . . . . . . . . 4
Configuring and Optimizing the Software Layer . . . . . . . . . . . . . . 5
Configuring and Optimizing the Hardware Layer . . . . . . . . . . . . . . 6
Benchmarking . . . . . . . . . . . . . . . . . . . 6
Conclusion . . . . . . . . . . . . . . . . . . . . . . 7

## Executive Summary

The amount of data being produced every day is growing at an astounding rate. The term "big data" has been coined to represent these new large and complex data sets. Traditional database management tools are no longer a good match for processing and managing big data. Fortunately, there are new tools available, like the Hadoop* framework, that are built to handle the challenge with ease.

This paper provides guidance for optimizing and tuning Intel® Distribution for Apache Hadoop* (Intel® Distribution) software, a big data system optimized to run on Intel processor-based architecture. This guidance is based on benchmark testing done both at Intel and at customer sites. It begins with an introduction to big data and the Intel Distribution software, and then breaks down the Hadoop system into its component layers. The guide then provides tips for hardware and software configuration, followed by tuning best practices that are geared toward providing optimal performance of the Intel Distribution based on the type of workload.

There are many players involved in configuring and managing a Hadoop environment. Throughout this guide, we've clearly identified which sections are of most interest to various roles in your IT organization.

## Introduction

Data is exploding at a phenomenal rate, with worldwide growth predicted to reach 8 zettabytes by 2015. Much of this data is characterized by data sets that are larger, more varied in structure and format, and generated at a faster rate than ever before—often referred to as big data. The analysis of big data presents new challenges for IT, but also exciting opportunities for organizations to gain richer insights into customers, partners, and their business.

The Hadoop platform was designed to solve the challenge of big data as well as complex data, such as a mixture of unstructured and structured data types. Although the Hadoop framework excels at processing and managing large data sets, there are many variables that should be fine-tuned to support each specific Hadoop environment for optimal performance.

Some Hadoop workloads will be CPU intensive, such as analytical jobs, while others will be I/O intensive, such as extract, transform, load (ETL) jobs. Configuration and tuning decisions within the Hadoop platform, including hardware and software, should be made based on the type of workload being performed. This optimization guide provides best-practice guidelines for a well-balanced Hadoop system based on testing performed with two HiBench benchmarking workloads, developed by Intel and now available as open-source software: TeraSort and WordCount.

HiBench is a comprehensive Hadoop benchmark suite developed by Intel and now made available as open-source software. Guidance about when to use the other 10 workloads in the suite can be found later in this paper.

## Components of Intel Distribution

The Hadoop system is composed of a number of components that are integrated in layers. The performance of any Hadoop system is based on optimal tuning of each layer. Because there are so many variables within several interdependent parts in each layer, tuning and optimizing a Hadoop system can be a challenge and may take some time. Also, each layer may be controlled by different IT teams, including developers and infrastructure teams, making the coordination of tuning efforts imperative.

Looking top down, the first tunable layer is the application layer. Any user applications developed for the Hadoop framework will have variables that can be set to provide the best performance for that application. The next layer is the actual Hadoop framework, which includes its two main components, MapReduce and the Hadoop* Distributed File System (HDFS*). Settings can be customized in the XML files found in the configuration folder.

The third layer is the software layer, which includes the Java* Virtual Machine (JVM*) as well as the C and C++ scripts, where parameters can be set accordingly. The fourth layer is the operating system, and the fifth layer is the hardware resources. Selection of hardware, such as CPU, memory, type of network interface card (NIC), and number and type of hard drives can greatly affect the performance of the Hadoop system.

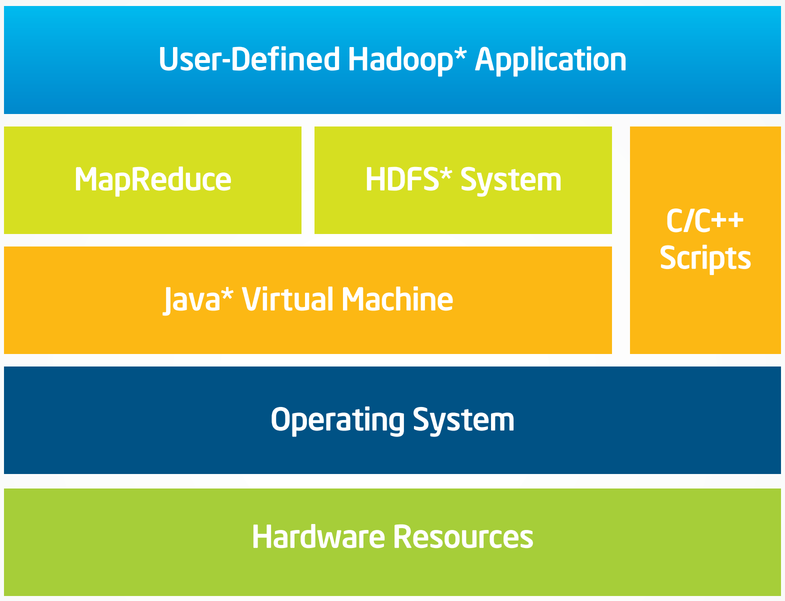

**Figure 1.** The layers of Intel® Distribution for Apache Hadoop* software.

## Resource Recommendations

Achieving optimal results with Intel Distribution starts with choosing the correct hardware and software stacks. The following hardware recommendations were developed to create optimal balance between performance and the total cost of ownership. The software recommendations were developed through extensive lab testing and represent best practices for performance, stability, and other benefits.

| Hardware Recommendations | |
|---|---|
| **Server platform** | ▪ Use a low-power server board, such as Intel's Server Board S5500WB, to increase electricity savings by 30% compared to the S5520UR model. |
| **Hard drives** | ▪ Native Command Queuing (NCQ)-supported, standard Serial AT Attachment (SATA) hard disk drives (HDDs) with 1–2 TB capacity per drive.<br>▪ 4–6 HDD JBOD for a balanced cluster.<br>▪ 12 or more JBOD for large-capacity, I/O-intensive requests.<br>▪ RAID is not recommended. Direct attached storage is preferable. |
| **Memory** | ▪ Each CPU core should have 1–2 GB of RAM.<br>▪ Select error correcting code (ECC) memory to detect and correct errors that may be introduced during the storage and transmission of data. |
| **Processor** | ▪ Select multicore boxes for better operational efficiency.<br>▪ Consider the cost vs. performance tradeoffs, as more cores will greatly increase the computing power. For example, a dual-socket, 12-core Intel® Xeon® processor 5600 performs about 20%–40% better than a dual-socket 8-core Intel Xeon processor 5500 for CPU-intensive workloads. |
| **Network interface card (NIC)** | ▪ NIC should support multiple receiving (RX) and transmitting (TX) queues.<br>▪ Adding more 1 GB Ethernet ports is more effective than increasing the port size, such as using one 10 GB network interface card, in a multicore system. |
| **Software Recommendations** | |
| **Operating system** | ▪ Linux* operating system. |
| **Hadoop* framework** | ▪ Hadoop framework version is 0.20.x or 1.0.x.<br>▪ Tests showed that version 0.19.1 was three times slower to finish the same job than version 0.20.1 and had two job- and shuffle-scheduling issues. |
| **Java* Virtual Machine (JVM*)** | ▪ Oracle Java HotSpot* VM (formerly Sun* HotSpot JVM).<br>▪ Benchmarking studies of short-duration tasks showed that Java HotSpot VM outperformed Oracle JRocket JVM. When considering the Java HotSpot VM, select 64-bit after version 1.6, update 14. |

## Optimizing and Tuning the Hadoop System

*This section is most relevant to application users and developers.*

The Hadoop framework solves the big data problem by managing petabytes or even larger amounts of data. Processing large amounts of data involves reading and writing activities within the Hadoop system. These activities are very resource intensive, so it is especially important to finely tune these activities as much as possible for best performance.

### Reduce disk and network I/O

Disk and network I/O activities can be reduced by tuning the memory buffer threshold and by using compression, which reduces the bandwidth and CPU needed for processing.

#### *MapReduce—Memory Buffer*

- ☐ *io.sort.factor* – This represents the number of input stream files to be merged at once during map or reduce tasks. The value should be sufficiently large (e.g., 100) rather than the default value of 10.
- ☐ *io.sort.mb* – This is the total size of the result and its metadata buffer and is associated with map tasks. The default value of 100 can be set higher, according to the HDFS block size, to 200 or 300.
- ☐ *io.sort.record.percent* – This is the percentage of the total metadata buffer size. The key-value pair combines the account information, which is fixed at 16 bytes, with the actual record, and is represented as a percentage ratio. This ratio should be adjusted based on the size of the key-value pair of the particular job, including the number of records and record size. Larger records should have a smaller account information ratio and smaller records should have a larger account information ratio.
- ☐ *mapred.job.shuffle.input.buffer.percent* – Increase the shuffle buffer size to store large and numerous map outputs in memory, while reserving part of the memory for the framework.
- ☐ *mapred.inmem.merge.threshold* – Avoid spill frequency by setting the value high enough, or set to zero (0) to disable it.
- ☐ *mapred.job.shuffle.merge.percent* – Adjust this value to balance between the spill frequency and the probability of the copier thread getting blocked.
- ☐ *mapred.job.reduce.input.buffer.percent* – Specify a relatively high value rather than the default (0), which can decrease disk I/O operations. Reserve enough memory space for the real reduce phase.

#### *MapReduce—Compression*

- ☐ *mapred.output.compress* or *mapred.compress.map.output* – Compression can be enabled (or disabled) to compress both the intermediate and final output data sets on HDFS system. These settings should be set to True.
- ☐ *mapred.output.compression.codec* or *mapred.map.output.compression.codec* – The codec can be configured using any one of a variety of compression types such as zlib, LZO, and gzip. Benchmark results indicate that LZO and Snappy are the most well-balanced and efficient codecs.

  Benchmarking tests compared using no compression to using zlib and LZO codecs. Although zlib has a higher compression ratio than LZO (meaning it saves more I/O bandwidth), it still takes longer to complete. Tests also found that zlib overwhelmed the CPUs. Tests showed that LZO functioned well across hardware resources, showing a 45 percent performance gain over zlib.
- ☐ *mapred.output.compression.type* – Each block in the HDFS system contains several records. Therefore, block-level compression is always better than record-level compression.

#### *HDFS System—Block Size and Handlers*

- ☐ *dfs.block.size* – By default, the minimum block file size is 64 MB. To increase performance and decrease the mapping time, this number should be increased to 128 MB or 256 MB. An increase from 128 MB to 256 MB reduced running time by 7 percent.

### Balance allocated resources

Carefully managed resources can improve load balancing and decrease the risk of failure.

#### *MapReduce—Load Balancing*

- ☐ *mapred.reduce.tasks* – Generally, the number of reduce tasks should be smaller than map tasks in a Hadoop job.
- ☐ *mapred.reduce.slowstart.completed.maps* – Depending on the job, this value can be increased or decreased to set the delay for the reduce tasks, which leaves resources available for other tasks. Use a higher value for larger data sets to increase the delay and smaller values for smaller data sets to decrease the delay. For example, if the variable is set to 50 percent, then the reduce tasks will start after 50 percent of the mapped tasks have finished.
- ☐ *mapred.reduce.parallel.copies* – This setting tracks the number of copies read that can be executed concurrently. The default setting is 5. To speed up the sort workload, specify a value between 16 and 25. Tests show that there is no benefit to setting the number much higher than this.

## Configuring and Optimizing the Software Layer

*This section is most relevant to the IT infrastructure team.*

Selecting and configuring the operating system and application software have major implications for performance and stability.

### Configure Java settings

The Hadoop framework and many of its applications run on Java. It is extremely important that the JVM runs as optimally as possible.

#### *Garbage Collection/Java Virtual Machine*

☐ Use "server" mode to appropriately manage resources for garbage collection (GC), especially for Hadoop processes such as JobTracker and NameNode.

☐ Enable the following GC options to support the Hadoop framework:

- Use parallel GC algorithm
- XX:ParallelGCThreads=8
- XX:+UseConcMarkSweepGC

☐ Set the parameter *java.net.preferIPv4Stack* to True to reduce overhead.

### Configure Hadoop framework settings

Settings for the HDFS system and MapReduce also need to be configured for optimal performance.

#### *HDFS System—Block Size and Handlers*

☐ *dfs.datanode.max.xcievers* – The default maximum number of threads that can connect to a data node simultaneously is 256. If this value is not high enough, you will receive an I/O exception error. To prevent this error, increase the xreceiver number to a higher number, such as 2,048, before starting the data node services.

#### *MapReduce—Load Balancing*

☐ *mapred.tasktracker.[map/reduce].tasks.maximum* – This setting determines the right number of task slots on each TaskTracker. The maximum number of map and reduce slots will be set in the range of (cores_per_node)/2 to 2x(cores_per_node). For example, an Intel® Xeon processor 5500 with eight cores, dual sockets, and hyper-threading (HT) turned on would require 16 map slots and eight reduce slots. For an Intel Xeon processor 5600 with 12 cores and dual sockets, the variable should be set to 24 map slots and eight reduce slots.

### Optimize Linux* operating system installation

The subsystem in the Linux operating system can be configured to improve I/O performance for the Hadoop system.

#### *File System*

☐ Use ext4 as the local file system on slave nodes. Although ext2, ext3, and XFS can be used, benchmarking studies show significant performance improvements with ext4. Gains were over 15 percent when using ext4 over the default, ext3, and even greater with other file system types.

☐ When using the ext4 file system, disable the recording of the file system access time, using *noatime* and *nodiratime* options, to improve performance as much as 10 percent.

☐ When using the ext4 file system, use the *ordered* or *writeback* journal mode for increased performance. Using journal mode in other file system types will actually decrease performance and should be disabled.

☐ When using the ext4 file system, increase the inode size from 256 K to 4 MB *(-T largefile4)* to improve performance by as much as 11 percent.

☐ Increase the read-ahead buffer size to improve the performance of sequential reads of large files. For example, increasing the size from 256 sectors to 2,048 sectors can save about 8 percent in running time.

#### *Operating System*

☐ Increase the Linux open-file-descriptor limit using */etc/security/limits.conf.* The default value of 1,024 is too low for the Hadoop daemon and may result in I/O exceptions in the JVM layer. Increase this value to approximately 64,000.

☐ If using kernel 2.6.28, increase the *epoll* file descriptor limit using */etc/sysctl.conf.* The default value of 128 is too low for the Hadoop daemon. Increase it to approximately 4,096.

☐ Increase *nrproc* (number of processes) in */etc/security/limit.conf.* The default value is 90, which is too small to run the Hadoop daemon and may result in I/O exception errors like "java.lang.OutOfMemoryError: unable to create new native thread." For the Red Hat* Enterprise Linux 6 operating system, edit *90-*.conf* under folder *etc/security/limits.d,* and set the hard and soft *nproc* to unlimited.

## Configuring and Optimizing the Hardware Layer

*This section is most relevant to the IT Infrastructure team.*

Determining the configuration of the servers in the cluster is critically important for providing the highest performance and reliability of these machines. Because workloads may be bound by I/O, memory, or processor resources, system-level hardware also may need to be adjusted on a case-by-case basis.

### Enable hardware settings

Optimizing hardware is often a balance between cost, capacity, and performance.

#### *Processor*

☐ Enabling hyper-threading will benefit CPU-intensive workloads and does not impact I/O-intensive workloads. For example, benchmarking tests show that HT can run as much as 25 percent more tasks per minute.

#### *Network*

☐ Enable channel bonding to resolve network-bound and I/O-intensive workloads. The channel bonding of two NIC ports will double the bandwidth and can improve the sort workload running time by 30 percent.

☐ Multiple-RX/TX-queue supported. Try to bind the queue to a different core, which can spread out the network interrupts onto different cores.

#### *Hard Drive Settings*

☐ Run in Advanced Host Controller Interface (AHCI) mode with Native Command Queuing (NCQ) enabled to improve the I/O performance of multiple, simultaneous read/write activities.

☐ For better I/O performance, enable the hard drive's write cache.

## Benchmarking

Benchmarking is the quantitative foundation for measuring the success of any computer system. Intel developed the HiBench suite as a comprehensive set of benchmarks for the Hadoop framework. The individual measures represent important Hadoop applications across a range of tasks. HiBench includes synthetic microbenchmarks as well as real-world Hadoop applications representative of a wider range of large-scale data analytics (for example, search indexing and machine learning).

Not all these benchmarks may be relevant for your organization. The following will help you understand what each benchmark measures, so you can map the relevant ones to your own Hadoop environment.

HiBench 2.1 is now available as open source under Apache* License 2.0 at https://github.com/hibench/HiBench-2.1.

**HiBench: The Details**
Intel's HiBench suite looks at 10 workoads in four categories.

| CATEGORY | WORKLOAD | INTRODUCTION |
|---|---|---|
| Micro benchmarks | Sort | ▪ This workload sorts its input data, which is generated using the Hadoop* RandomTextWriter example.<br>▪ Representative of real-world MapReduce jobs that transform data from one format to another. |
| | WordCount | ▪ This workload counts the occurrence of each word in the input data, which is generated using Hadoop RandomTextWriter.<br>▪ Representative of real-world MapReduce jobs that extract a small amount of interesting data from a large data set. |
| | TeraSort | ▪ A standard benchmark for large-size data sorting that is generated by the TeraGen program. |
| | Enhanced DFSIO | ▪ Tests HDFS* system throughput of a Hadoop cluster.<br>▪ Computes the aggregated bandwidth by sampling the number of bytes read or written at fixed time intervals in each map task. |
| Web search | Nutch Indexing | ▪ This workload tests the indexing subsystem in Nutch, a popular Apache* open-source search engine. The crawler subsystem in Nutch is used to crawl an in-house Wikipedia* mirror and generates 8.4 GB of compressed data total (for about 2.4 million web pages) as workload input.<br>▪ This large-scale indexing system is one of the most significant uses of MapReduce (for example, in Google* and Facebook* platforms). |
| | Page Rank | ▪ This workload is an open-source implementation of the page-rank algorithm, a link-analysis algorithm used widely in Web search engines. |

**HiBench: The Details (continued)**
Intel's HiBench suite looks at 10 workoads in four categories.

| CATEGORY | WORKLOAD | INTRODUCTION |
|---|---|---|
| Machine learning | K-Means Clustering | ▪ Typical application area of MapReduce for large-scale data mining and machine learning (for example, in Google and Facebook platforms).<br>▪ K-Means is a well-known clustering algorithm. |
| | Bayesian Classification | ▪ Typical application area of MapReduce for large-scale data mining and machine learning (for example, in Google and Facebook platforms).<br>▪ This workload tests the naive Bayesian (a well-known classification algorithm for knowledge discovery and data mining) trainer in the Mahout* open-source machine-learning library from Apache. |
| Analytical query | Hive* Join | ▪ This workload models complex analytic queries of structured (relational) tables by computing the sum of each group over a single read-only table. |
| | Hive Aggregation | ▪ This workload models complex analytic queries of structured (relational) tables by computing both the average and sum for each group by joining two different tables. |

## Conclusion

Fine-tuning the environment for specific workloads calls for a fairly in-depth analysis, which can take time and should be done on a case-by-case basis. It's important to first analyze the workloads to determine whether they are I/O bound or CPU bound. Most Hadoop applications are data intensive, so start by tuning the I/O-related application variables first, such as adding compression. For the CPU-bound workloads, optimizing may involve adding more hardware, such as more cores or nodes.

Requirements for every enterprise Hadoop system will vary. Sometimes the requirements will even vary within the enterprise, depending on the job or workload. Fortunately, there are many different variables within the Hadoop system component layers that can be adjusted to provide the best performance for the workload. It may take some time for tuning to reach an optimal state, but the up-front time spent will pay off in the long run in terms of performance and, ultimately, total cost of ownership for the Hadoop environment.

To find out more about the Intel Distribution for Hadoop software and partner program, visit **intel.com/bigdata**.

Contact Intel at:
Phone: 1-855-229-5580
E-mail: **asipcustomercare@intel.com**

[1] "Big Data Infographic and Gartner 2012 Top 10 Strategic Tech Trends." *Business Analytics 3.0* (blog) (November 11, 2011). http://practicalanalytics.wordpress.com/2011/11/11/big-data-infographic-and-gartner-2012-top-10-strategic-tech-trends/

[2] Huang, Shengsheng, Jie Huang, Jinquan Dai, Tao Xie, Bo Huang. *The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis.* IEEE (March 2010).