This Dell Technical White Paper explains how to tune BIOS, OS and Hadoop settings to increase performance in Hadoop workloads.

Donald Russell
Solutions Performance Analysis
# Tuning Hadoop on Dell PowerEdge Servers

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executive summary</td>
<td>5</td>
</tr>
<tr>
<td>Introduction</td>
<td>5</td>
</tr>
<tr>
<td>Key findings</td>
<td>5</td>
</tr>
<tr>
<td>Compression</td>
<td>5</td>
</tr>
<tr>
<td>Block size</td>
<td>5</td>
</tr>
<tr>
<td>Number of hard drives</td>
<td>5</td>
</tr>
<tr>
<td>File system</td>
<td>5</td>
</tr>
<tr>
<td>OS settings</td>
<td>5</td>
</tr>
<tr>
<td>Java settings</td>
<td>5</td>
</tr>
<tr>
<td>Hadoop settings</td>
<td>6</td>
</tr>
<tr>
<td>Methodology</td>
<td>7</td>
</tr>
<tr>
<td>Hadoop cluster configuration</td>
<td>7</td>
</tr>
<tr>
<td>BIOS</td>
<td>7</td>
</tr>
<tr>
<td>Power management</td>
<td>7</td>
</tr>
<tr>
<td>ASPM</td>
<td>7</td>
</tr>
<tr>
<td>Raid controller or SATA power management</td>
<td>8</td>
</tr>
<tr>
<td>Summary</td>
<td>8</td>
</tr>
<tr>
<td>OS and file system</td>
<td>8</td>
</tr>
<tr>
<td>Open file descriptors and files</td>
<td>8</td>
</tr>
<tr>
<td>File system</td>
<td>9</td>
</tr>
<tr>
<td>Network</td>
<td>10</td>
</tr>
<tr>
<td>Huge pages</td>
<td>12</td>
</tr>
<tr>
<td>Linux kernel swappiness parameter</td>
<td>13</td>
</tr>
<tr>
<td>Linux IO scheduler</td>
<td>13</td>
</tr>
<tr>
<td>Summary</td>
<td>14</td>
</tr>
<tr>
<td>Hadoop settings</td>
<td>15</td>
</tr>
<tr>
<td>Hard drive scaling</td>
<td>15</td>
</tr>
<tr>
<td>Hadoop file system compression</td>
<td>16</td>
</tr>
<tr>
<td>HDFS block size</td>
<td>17</td>
</tr>
<tr>
<td>MapReduce</td>
<td>19</td>
</tr>
<tr>
<td>The MapReduce process</td>
<td>20</td>
</tr>
<tr>
<td>Map process</td>
<td>21</td>
</tr>
<tr>
<td>Reduce process</td>
<td>22</td>
</tr>
</tbody>
</table>
Executive summary

Introduction

Hadoop is a Java-based distributed framework designed to work with applications implemented using MapReduce modeling. This distributed framework makes it possible to pass the load on to thousands of nodes across the whole Hadoop cluster. The nature of distributed framework also allows for node failure without cluster failure. The Hadoop market is predicted to grow at a compound annual growth rate over the next several years. Several good tools and guides describe how to deploy Hadoop clusters, but very little documentation tells how to increase performance on a Hadoop cluster once it is deployed. This white paper explains several BIOS, OS, file system, Hadoop and Java tunings that can increase performance of a Hadoop cluster. These performance tunings require changes to the BIOS, OS, Hadoop and JVM configuration files.

Key findings

Compression

Compression increases performance by up to 20 percent under certain Hadoop activities.

Block size

Changing the Hadoop Distributed File System (HDFS) block size decreases the overhead of the MapReduce task workload. If it is set to match the optimized block size of the controller on the Hadoop data nodes, it can increase Hadoop MapReduce performance by up to 21 percent.

Number of hard drives

The number of hard drives (HDD) attached to the data nodes can decrease the Hadoop job execution time by up to 20 percent for each HDD added, until storage-bus saturation is achieved.

File system

Changing certain file system settings in the native OS increased Hadoop performance by up to 10 percent.

OS settings

Changing certain default OS settings can increase Hadoop performance by up to 15 percent and increase the chance of completion of the Hadoop job.

Java settings

By adding certain Java options to the Hadoop JVMs, Hadoop performance can increase by up to five percent.
Hadoop settings

By changing certain Hadoop parameters, performance can increase by up to 12 percent in MapReduce jobs.
Methodology

Recommendations presented in this white paper are based on experience with setting up and tuning Hadoop workloads on multiple-cluster configurations. This white paper approaches tuning Hadoop clusters systematically, and explores each option and the differences it makes to overall performance. In this white paper, all performance data was created by using 1TB Teragen \ Terasort workloads. The performance achieved depends on the Hadoop workload that is implemented on your Hadoop cluster.

This white paper tries to explain the recommendations and the tradeoffs associated with the changed settings, so readers can better determine which settings to change to benefit their Hadoop cluster.

Hadoop cluster configuration

The Primary Hadoop cluster used to test most of the tunings in this white paper had the following configuration.

- Three PowerEdge R620 rack servers, with two Intel® Xeon® E5-2680 processors, 64GB of RAM, eight 2.5” 300GB 10k SAS in R10 (Master / Secondary/ Edge / HA)
- Four PowerEdge C8220 double-width compute (DWC), with two Intel Xeon E5-2640 processors, 64GB of RAM, 12 2.5” 1TB 7.2 SATA in JBOD (data nodes)

A secondary, or test-bed, cluster had the following configuration.

- 3 R410 2 Xeon 5620, 32GB Ram, four 3.5” 160GB 7.2K SATA R10 (Master/Secondary/HA/Edge)
- 4 R410 2 Xeon 5620, 32GB Ram, four 3.5” 160GB 7.2K SATA in JBOD (data nodes)

BIOS

This section explores and explains BIOS settings so that their benefits can be better understood. The BIOS option can achieve a performance gain of a few percent to over 10 percent, depending on the Hadoop workload and the default BIOS settings of the system.

Power management

In most cases BIOS power management set to Max Performance results in better performance in Hadoop workloads than does OS control or Balance. This is because the BIOS turns off most if not all of the power-saving features. A few watts of extra power draw on one system might not seem like a lot, but across thousands of Hadoop nodes it might be too much. The extra power usage must be weighed against the performance gain achieved.

ASPM

For Hadoop performance benefits this BIOS option must be turned off or disabled. In some Dell BIOS the ASPM links are broken up into separate PCI Express® (PCIe) link options. If this is the case, disable or turn off ASPM settings for each device in the Hadoop node that is being used for the cluster. If you
need to save power on the cluster, look at the following information to determine whether or not to disable these BIOS features.

While ASPM reduces power consumption, it can also result in increased latency because the serial bus needs to be woken up from low-power mode, possibly reconfigured, and the host-to-device link re-established. This is known as ASPM exit latency.

Currently there are two low-power modes specified by the PCIe 2.0 specification which are L0s and L1 mode. The L0s mode sets low-power mode only for the serial link, downstream of the serial controller in one direction. The L1 mode results in greater power reduction and is bidirectional, but it comes with the penalty of greater exit latency. This BIOS option slows the throughput to any device on which it is enabled.

Raid controller or SATA power management

The power management for the RAID or onboard SATA controller can help reduce node power consumption, but comes at the cost of storage performance. Turning off the power management for the SATA controller in the System BIOS or the RAID controller firmware can increase Hadoop node performance. These options and performance gains differ from system to system, but are worth exploring.

Summary

Explore the Hadoop nodes BIOS options, especially the power management options. Turning off the BIOS power management options increase overall system performance and Hadoop performance.

OS and file system

By changing some default Linux settings, Hadoop performance can increase by 15 percent, a side benefit of some of these changes is that Hadoop workloads will complete without failing. In this section, we will discuss the changes and their benefits.

Open file descriptors and files

For most Hadoop workloads, changing the default maximum number of open file descriptors will help with Hadoop tasks completing without errors. The reason that some Hadoop jobs fail or seem to get stuck is because the default setting of 1024 open file descriptors per process is too low. To correct this issue, setting the default maximum open file descriptors to 32832 is a good idea. This will allow Hadoop enough headroom to write all the files it needs to create.

To verify the current max open file descriptors use the following command.

For RedHat or Centos (as root)

```
# cat /proc/sys/fs/file-max
```

For SLES or Debian-based Linux distributions like Ubuntu

```
$ sudo cat /proc/sys/fs/file-max
```
To increase the max open file descriptors for Hadoop, run the following commands for the HDFS and Hadoop users.

For RedHat or CentOS (as root)

# cat /proc/sys/fs/file-max

For SLES or Debian-based Linux distributions like Ubuntu

$ sudo cat /proc/sys/fs/file-max

To increase the max open file descriptors for Hadoop, run the following commands for the HDFS and Hadoop users.

For RedHat or CentOS (as root)

# su -(hdfs & Hadoop users)
# ulimit -S 4096
# ulimit -H 32832

For SLES or Debian-based Linux distributions like Ubuntu

$ sudo su -(hdfs & Hadoop users)
$ ulimit -S 4096
$ ulimit -H 32832

If you run into the Linux error that tells you that you have reached the max number of files, use the following command to increase the amount of system wide file descriptors.

For RedHat or CentOS (as root)

# sysctl -w fs.file-max=100000

For SLES or Debian-based Linux distributions like Ubuntu

$ sudo sysctl -w fs.file-max=100000

Now that you have created a new kernel variable, add it to sysctl to make it permanent.

For RedHat or CentOS (as root)

# vi /etc/sysctl and append the file with the fs.file-max = 100000

For SLES or Debian based Linux distributions like Ubuntu

$ sudo vi /etc/sysctl and append the file with the fs.file-max = 100000

Now, to verify that the setting took hold, log out and log in or reboot the system. Once logged in, check everything with the following commands.

For RedHat or CentOS (as root)

# cat /proc/sys/fs/file-max

For SLES or Debian-based Linux distributions like Ubuntu

$ sudo cat /proc/sys/fs/file-max

File system

Linux distributions can have different default file system preferences. After testing different Linux file systems, it has been found that EXT4 seems to be better than EXT3. The new features in EXT4, like
multiblock allocation and delayed allocation, increase the performance of the EXT4 over EXT3. To understand this better, let’s explore the differences of how EXT3 writes blocks to the HDD. When a file is created or data inserted into an existing file, EXT3 calls the block allocator, once for each block. If there are multiple concurrent writes, files can easily become fragmented. On the other hand, EXT4 buffers the data by using delayed allocation and allocates groups of blocks. This allows the multiblock allocator to better lay the data contiguously on disk. Contiguous files are easy for mechanical hard drives to read, which increase the overall performance of storage IO.

Adding the file system mount option of noatime, which also covers nodiratime, to the Hadoop data node’s hard drives, will reduce the amount of writes to the HDDs and increase performance and decrease fluctuations in Hadoop performance results. This is because each file touched or directory created does not get its date stamp updated or created. In Hadoop, many intermediate files are created that will only live while the Hadoop job is being run and will never be seen by the end user. There is no need to have access or create dates stamps on these files. Below is an example of what the HDFS mount point may look like.

```
#vi /etc/fstab
/dev/sdb1 /mnt/hdfs/1 ext4 noatime 0 0
/dev/sdc1 /mnt/hdfs/2 ext4 noatime 0 0
/dev/sdd1 /mnt/hdfs/3 ext4 noatime 0 0
/dev/sde1 /mnt/hdfs/4 ext4 noatime 0 0
```

Another optimization for the data nodes is to save space by tuning the reserve blocks within the file system. The reserve block setting is to reserve enough file space on the file system so the root account can log into the system. By default, Linux sets the reserve block to five percent. When this setting was first created hard drives were smaller and five percent was the correct size, but now on 1TB hard drives that is a lot of wasted space, around 50GB. To check the reserved block count, use the following command `tune2fs -l /dev/sdaX` and compare it to the block count. It is safe to assume that if one partition is set this way all the partitions on the hard drives in the data nodes are set this way. To change the reserved block count to one percent, use the following command, `tune2fs -m 1 /dev/sdaX`. This command will have to be run on each of the partitions on the hard drives in the data node. Just to give you example, on the Hadoop cluster set up for this white paper, the data nodes were four Dell PowerEdge C8220 with 12 1TB hard drives. By changing the reserve blocks to one percent the HDFS space increased by 1.6 TB. Had the reserve block been set to zero percent on the data partitions of the HDFS, the increase in space would have been over 2TB, which is a lot of space on a small cluster. You can set it to zero percent, if needed, on the data partitions of the data nodes, because the root account log in files are generally created on the root partition by default. To be safe, keep the default settings for your system partitions and disks. This prevents the root account from getting locked out.

**Network**

Two network-related settings can affect Hadoop performance issues and failures. Changing the `net.core.somaxconn` Linux Kernel settings from the default of 128 to 1024 helps with burst requests from the Namenode and Jobtracker. This option sets the size of the listening queue, or the number of connections that the server can setup at one time. Verifying that the `txqueuelen` is set to 4096 or higher changes the transmit queue length to better accommodate the burst traffic seen in the network traffic of Hadoop clusters.
The following commands change these network settings.

For RedHat or Centos (as root):
   # sysctl -w net.core.somaxconn=1024

For SLES or Debian-based Linux distributions like Ubuntu:
   $ sudo sysctl -w net.core.somaxconn=1024

After you have created a new Kernel variable, add it to `sysctl` to make it permanent.

For RedHat or Centos (as root):
   # vi /etc/sysctl

For SLES or Debian-based Linux distributions like Ubuntu
   $ sudo vi /etc/sysctl

Append the file with the net.core.somaxconn=1024.

To verify that the setting took hold, log out and log in or reboot the system. Once logged in, check everything with the following commands.

For RedHat or Centos (as root):
   # sysctl net.core.somaxconn

For SLES or Debian-based Linux distributions like Ubuntu:
   $ sudo sysctl -a | grep somaxconn

To change the `txqueuelen` use the following command.

For RedHat or Centos (as root):
   # ifconfig eth# txqueuelen 4096

For SLES or Debian-based Linux distributions like Ubuntu:
   $ sudo ifconfig eth# txqueuelen 4096

To make this change happen automatically, you can add this command to the `rc.local` file of the system.

A couple of other network performance tweaks, which are more experimental, can increase Hadoop performance. These options increase the read and write cache sizes for the network stack. All of these options can be tested with the `sysctl -w` command or made permanent by adding the variable to the `/etc/sysctl` file.

- increase TCP max buffer size
  net.core.rmem_max = 134217728 (128MB)

- increase Linux autotuning TCP buffer limit
  net.ipv4.tcp_rmem = 4096 87380 134217728
Huge pages

The Linux feature _Transparent Huge Pages_ increases performance in a wide range of applications, including Hadoop workloads, but, a sub feature of Transparent Huge Pages called _Compaction_ can cause issues with Hadoop workloads. During Hadoop benchmark testing with the Compaction option on, there was up to a 25 percent fluctuation in results. Once the Compaction option was disabled, this fluctuation stopped. When it defragments the memory, the Transparent Huge Pages Compaction option creates high processor usage. This helps to better optimize Transparent Huge Page performance, but it steals processor resources during the defragmenting process, adversely affecting the performance of running tasks. On a normal server, most tasks complete within minutes, but some Hadoop jobs or tasks can run for days, depending on the size of the job and cluster. Any job or task that creates high processor utilization must be minimized.

Tuning the Transparent Huge Page Compaction

Most Hadoop workloads interact poorly with Transparent Huge Page Compaction. This paper shows how to minimize its effects by disabling it. To verify that the problem exists, use system monitoring tools, such as Top. If system processor usage is high — above 30 percent — during the Hadoop workload and Transparent Huge Pages are enabled, then the system is probably experiencing the Compaction problem.

Transparent Huge Page Compaction is generally enabled in all Linux distributions. To verify this, run the following command.

For RedHat or Centos:

```
# cat /sys/kernel/mm/redhat_transparent_hugepages/defrag
```

For SLES or Debian-based Linux distributions like Ubuntu:

```
$ sudo cat /sys/kernel/mm/tranparnet_hugepages/defrag
```

Transparent Huge Pages is good for overall performance. Defragging the memory increases the performance of Transparent Huge Pages, but the memory should not be defragmented while a Hadoop job is running. This paper recommends disabling the Compaction feature during the Hadoop jobs. This can be done by either echoing never or echoing always into the following commands below.

For RedHat or Centos:

```
# echo never > /sys/kernel/mm/redhat_transparent_hugepages/defrag
```

For SLES or Debian-based Linux distributions like Ubuntu:

```
$ sudo echo never > /sys/kernel/mm/tranparnet_hugepages/defrag
```

This command disables the Compaction feature of Transparent Huge Pages.
Linux kernel swappiness parameter

Any process that writes to the hard drives can decrease Hadoop performance. The Linux Kernel process called `vm.swappiness` checks for unused memory pages and swaps them out to the hard drive. The default value of `vm.swappiness` in most Linux distributions is 60. It can be set from 0 to 100. For Hadoop purposes, setting it to 0 is a good idea. This does not totally disable the feature. Linux still swaps to avoid out-of-memory states, but because this process swaps to disk even if there is plenty of free memory available, setting it to 0 reduces memory and disk latency. To verify the current setting, to change it or making it permanent, use the following commands.

Verify:
For RedHat or Centos:
```bash
# cat /proc/sys/vm/swappiness
```

For SLES or Debian-based Linux distributions like Ubuntu:
```bash
$ sudo cat /proc/sys/vm/swappiness
```

Try:
For RedHat or Centos:
```bash
# sysctl -w vm.swappiness=0
```

For SLES or Debian-based Linux distributions like Ubuntu:
```bash
$ sudo sysctl -w vm.swappiness=0
```

Making permanent:
For RedHat or Centos (as root):
```bash
# vi /etc/sysctl and append the file with the vm.swappiness=0
```

For SLES or Debian-based Linux distributions like Ubuntu:
```bash
$ sudo vi /etc/sysctl and append the file with the vm.swappiness=0
```

Linux IO scheduler

Changing the IO scheduler to Completely Fair Queuing (CFQ) provided up to a 15 percent improvement in performance. Some Linux distributions already use CFQ as their IO scheduler by default. Depending on storage hardware configuration, IO patterns, the requirements of the Hadoop application, determines which IO scheduler may perform better. If the Hadoop nodes have SSD’s or if there is a RAID controller in the node, to which the hard drives are attached, then NOOP can be a better choice. Because the NOOP scheduler lets the device take care of all of the IO scheduling within Linux. Linux has four IO schedulers — noo, anticipatory, deadline and cfq. To check which scheduler is being used and to change to a different one, use the following commands.

```bash
# or $ cat /sys/block/sda/queue/scheduler
```

To change the scheduler use the following commands, it must be set on all the hard drives in the Hadoop node.

For RedHat or Centos (as root):
```bash
# echo cfq > /sys/block/sda/queue/scheduler
```
For SLES or Debian-based Linux distributions like Ubuntu:

$ sudo echo cfq > /sys/block/sda/queue/scheduler

Summary

Default OS settings can affect the completion as well as the performance of Hadoop workloads. At the very least, change or verify the OS settings that cause the Hadoop job failures explained in this section. Then explore the performance benefits for the rest of the settings discussed.
Tuning Hadoop on Dell PowerEdge Servers

Hadoop settings

Depending on the system configuration of the Hadoop nodes, a couple of Hadoop settings can reduce the number of bottlenecks in the system, while running Hadoop workloads. The goal is to maximize the system resources used on the system. First, increase the Java heap sizes to help with overall Hadoop performance, but keep the Java Heap sizes within the amount of memory installed in the system. Second, keep the amount of Hadoop-spawned tasks tied to the number of processor cores installed in the system. The following configuration options can be found in the mapred-site.xml file. These parameters help tune the number of Hadoop tasks started and control the amount of memory used by Hadoop.

- mapred.reduce.tasks
- mapred.tasktrackers.map.tasks.maximum
- mapred.map.tasks
- mapred.tasktracker.reduce.task.maximum
- mapred.map.child.java.opts
- mapred.reduce.child.java.opts

Another good rule to follow is to have one reducer and two map tasks per processor core. If the system has enough memory, set the maximum Java heap size to 1GB or greater. Remember that three Java tasks will be running, so the JVMs for Hadoop will be reserving at least 3GB of system memory. Make sure you have more than 3GB of memory per processor core for all the processor cores installed in the system. Dedicating this much memory to the Hadoop JVMs keeps the MapReduce process in memory most of the time, which decreases the time it takes to complete the MapReduce requests.

Hard drive scaling

The number of Hard Drives attached to the Hadoop data nodes can increase performance. Putting the data on more than one hard drive allows the Hadoop workload to fetch more HDFS blocks from different hard drives, removing the bottleneck of getting the data from the disk. The Hadoop workload effectively acts like a raid controller and fetches the HDFS blocks from multiple locations at the same time. On its own this is not a tweak, but it can increase the overall performance of Hadoop workloads.

From the benchmarking performed, more hard drives installed in the data nodes increases Hadoop workload performance until storage bus saturation is achieved. This issue can be addressed with another storage controller, until no more hard drives can be added to the system or until the system bus reaches saturation.

The system used for benchmarking was a Dell PowerEdge C8220 DWC, which was able to take up to 12 hard drives. This system achieved up to a 79 percent decrease in normalized execution time, by adding the maximum number of hard drives to the system. Figure 1 shows an example of controller saturation and normalized execution time versus hard drives in Hadoop data nodes. Going from four to five disks gains a 20 percent decrease in the time taken to execute the Hadoop workload. By adding more disks until we reached 12 hard drives, the normalized execution time was decreased by 79 percent. The last hard drive gained only around a five percent decrease in normalized execution time, indicating that the storage controller is reaching bus saturation. To verify bus saturation, use TOP on the data nodes while a Hadoop job is running and check the IO wait during the map phase of the job.
When the storage bus is saturated, the IO wait will be above 30 percent. Keep this in mind when adding Hard drives to the system.

![Normalized execution time versus hard drives in Hadoop data nodes](image)

**Figure 1.** Normalized execution time versus hard drives in Hadoop data nodes

**Hadoop file system compression**

Using compression in Hadoop can help with cluster performance by reducing network and disk IO. It also reduces the disk space needed to store files, but the reduction comes at the expense of the processor time. Consider this performance tradeoff when picking the processors for the Hadoop data nodes. Hadoop has compression support for intermediate Map output data, input data and Reduce output data.

Hadoop also supports compression at the application level within the Java job itself. The support codec are Deflate, gzip, bzip2 and LZO (see Table 1). Each of these codecs has its own strengths and weaknesses. This white paper does not discuss application compression, input data compression and Reduce output data, because the benchmark used — Terasort — does not support compression in these phases of Hadoop.

Appendix A — LZO installation process compares no compression and LZO compression and shows the benefits of compression in the underlying layers of MapReduce (see Figure 2).
Explaining benefits of compression in the underlying layers of MapReduce might help with understanding of why compression gains performance in MapReduce jobs. Intermediate MapReduce output is data that is stored before a map reducer works on it never leaves the MapReduce process; it is only seen by the reducer. Because the MapReduce mapper and reducers might not be on the same system, compressing this data can save network and disk IO. LZO is generally not processor intensive, taking around 20 percent to 40 percent of processor capacity. Leaving enough processor overhead for other Hadoop tasks running on the system. LZO can be made to be splittable by adding the correct codec entries to the Hadoop configuration files or application code. All of settings changed to test compression are found in the mapred-site.xml file. The following properties were changed mapred.compress.map.output, mapred.map.output.compression.codec, mapred.output.compress, mapred.output.compression.type and mapred.output.compression.codec.

LZO compression can also help to break large files into manageable chunks by splitting and indexing the files. When using compression, it is advisable to use only compression formats that support splitting and indexing of the whole file, otherwise loss of locality can make MapReduce applications very inefficient. Compression can reduce normalized execution time for Hadoop jobs by up to 20 percent compared to when no compression is used.

### Table 1. Compression formats supported in Hadoop

<table>
<thead>
<tr>
<th>Compression format</th>
<th>Algorithm</th>
<th>Multiple files</th>
<th>Splittable</th>
<th>Java implementation</th>
<th>Native implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deflate</td>
<td>DEFLATE</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Gzip</td>
<td>DEFLATE</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Bzip2</td>
<td>bzip2</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>LZO</td>
<td>LZO</td>
<td>No</td>
<td>Yes (if codecs are installed)</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### HDFS block size

Changing the default HDFS block size can increase performance of a Hadoop cluster in two ways. First, it can better match the optimal block size of the controller to which the hard drives are attached on the data nodes. This increases performance of the IO to and from the hard drives and storage controller. Second, the HDFS block size affects the number of MapReduce tasks created for a MapReduce job. For example, with the default Hadoop HDFS block size of 64MB, a 1TB workload turns into 16000 MapReduce tasks. If the Hadoop HDFS block is changed to 384MB, that 1TB Hadoop Workload turns into 2652 MapReduce tasks.
This HDFS block size also optimized block size for the Dell RAID controller in the C8220 data nodes in the Hadoop benchmarking cluster. A 384MB HDFS block size gave the Hadoop cluster data nodes a boost on the IO side and decreased the MapReduce tasks by 83 percent, netting the system a 21 percent decrease in normalized time.

![Normalized job completion time](image)

**Figure 3.** Block size improvement in normalized job completion time

Note that the HDFS Block size does interact with another Hadoop parameter called the *io.sort.mb* (*hdfs-site.xml*). Table 2 shows that the *io.sort.mb* determines the amount of memory that can be used as a buffer for the IO sort process.

**Table 2.** HDFS block size and MapReduce tasks — change needed for *io.sort.mb* parameter

<table>
<thead>
<tr>
<th>1TB MapReduce Job</th>
<th>Comes from JAVA JVM memory</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HDFS Block Size in MB</strong></td>
<td><strong>Map tasks for Mapreduce</strong></td>
<td><strong>io.sort.mb</strong></td>
</tr>
<tr>
<td>64MB HDFS Block size</td>
<td>16000</td>
<td>100</td>
</tr>
<tr>
<td>128MB HDFS Block size</td>
<td>8000</td>
<td>192</td>
</tr>
<tr>
<td>256MB HDFS Block size</td>
<td>4000</td>
<td>320</td>
</tr>
<tr>
<td>386MB HDFS Block size</td>
<td>2652</td>
<td>450</td>
</tr>
<tr>
<td>512MB HDFS Block size</td>
<td>2000</td>
<td>576</td>
</tr>
<tr>
<td>640MB HDFS Block size</td>
<td>1600</td>
<td>704</td>
</tr>
<tr>
<td>768MB HDFS Block size</td>
<td>1333</td>
<td>832</td>
</tr>
</tbody>
</table>

Map tasks start with a function called an input split, which is based on three parameters. The *mapred.min.split.size*, *mapred.max.split.size* (both in *mapred-site.xml*) and *dfs.block.size* (*hdfs-site.xml*) configuration parameters decide the size of the input split. The total input data size and the input split size determine the total number of MapReduce tasks spawned by the Hadoop framework. Use Teragen and Terasort, while changing HDFS block size, which changes the input split and decreases...
the number of MapReduce tasks created until the optimal block size for the Hadoop data nodes is found. In some cases using an even larger *dfs.block.size* than the optimal block-size setting can decrease normalized time. By reducing the number of MapReduce tasks, this larger optimal block-size setting reduces some of the overhead of starting and tearing down of the JVMs for the MapReduce tasks. It also reduces the cost of merging map output segments in the Reduce phase, because the increased HDFS block size reduces the number of Map tasks. Rather than running large numbers of small MapReduce tasks, with all that overhead, it is better to run a smaller number of larger and longer MapReduce tasks. With larger HDFS block sizes the *io.sort.mb*, *io.sort.spill.percent* and *io.sort.record.percent* parameters must be adjusted to prevent Map output spills. See Figure 4.

For example, for a HDFS block size setting of 384MB, where each record is 100 bytes long, setting the *io.sort.mb* to 450MB, *io.sort.record.percent* to 0.21 and *io.sort.spill.percent* to 0.99 completely eliminates Map-side spills in the benchmarking cluster. There was a 3.17 percent decrease in normalized time, over what it took when Map-spills were happening during the Map process. This decrease in normalized time should be greater on larger Hadoop clusters.

![MapReduce process and parameter](image)

Figure 4. MapReduce process and parameter

Map output spills are written to disk, which increases MapReduce time by inducing disk IO latency. It is better to keep as much of the MapReduce process as possible in memory.

The *io.file.buffer.size* parameter (*core-site.xml*) also helps with IO latency. Setting this parameter to 131072 quadruples the default file buffer and decreases disk IO latency when reading and writing files. As the Map output is merged and written to HDFS, the *io.sort.factor* parameter (*hdfs-site.xml*) determines the number of streams feed into the merger. On a job with many map tasks increasing the *io.sort.factor* value from the default of 10 to something higher depends on the amount of disk spills you are seeing. By tuning all of these HDFS settings to the hardware capabilities of the Hadoop data nodes, more of the MapReduce process can be kept in memory, which decreases the overall time to complete the MapReduce tasks that are running.

**MapReduce**

MapReduce is more complicated to configure and benchmark than is HDFS. Understanding all the MapReduce parameters is important, so that settings can be safely changed and the performance increased.
The MapReduce process

A MapReduce job comprises two processes, or phases — the Mapping phase and the Reduce phase.

The Mapping phase distributes the job or data to all the data nodes in the cluster. Many Mapping processes can be created on each data node. To achieve the maximize performance in the Mapping phase and to take advantage of the parallelism of the Mapping process, use all the processor cores on the data nodes.

A MapReduce job goes through the following steps to complete a job. This explains at a very high level what is going on in the MapReduce process of Hadoop.

Figure 5. MapReduce job process

Figure 6. MapReduce work flow
The input splits determine the number of Mapping and Reduce processes to be created on the data nodes. Each Mapping process maps input data, sorts it into data groups with keys and writes it to HDFS partition, where the Reducer starts to process the map output.

All the data for a given key must be seen by the Reducer. In a cluster with a single Reducer, all the output from each Map is sent to the node with the Reducer. All the data and keys are merged from the Mapping process before the Reducer starts.

For a Hadoop cluster with multiple Reducers, the Mapping output is partitioned, creating one partition for each Reducer. These partitions contain all the data and keys needed by a Reducers. These partitions are then sent over then network to each of the Reducers in the cluster.

To minimize the amount of data that to be transferred, and to speed the execution time, keep the data from the Mapping process and the Reducers on the same data node.

**Map process**

The Mapping phase starts with the input split, which is discussed in the HDFS section of this paper. While the Map process is writing the input split data to the HDFS partitions, it uses a memory buffer called the `io.sort.mb` *(hdfs-site.xml)*. By default this is set to 100MB. If the HDFS block size is increased, this memory buffer must also be increased. Note that the `io.sort.mb` memory buffer is taken from the JVM memory reserved for the Map process.

Another setting that coexists with the `io.sort.mb` is the `io.sort.record.percent`; this is a portion of memory within the `io.sort.mb` memory that is used for tracking record boundaries. By default the `io.sort.record.percent` parameter is set to .05, or five percent. Each tracking record is 16 bytes, so if both of these setting are left at default settings for Hadoop, the maximum Mapping process output cannot be greater than 327680. The `io.sort.mb` memory buffer fills until it reaches 80 percent, which is the default setting for `io.sort.spill.percent`. At this point to prevent buffer overflow the buffer is flushed to disk. This happens at 80 percent so the Map process can continue running and spawn a separate thread to handle the spill, so the memory buffer is not completely filled up before the Mapping process can complete. If the Mapping process outputs are too large for the `io.sort.mb` memory buffer, excessive spills to disk increase the overall Hadoop job execution time. Increasing these parameters — giving them more memory and allowing the map process to stay in memory — can decrease the execution time. The main reason that all the disk spills increase execution time is because of disk IO; they also have to be merged back into sorted files before the Reduce process can use them. At this point all of the merged files from the Mapping phase are passed to the Reduce phase of the MapReduce process.
Reduce process

The Reduce phase consists of five different steps — copy, shuffle, merge, sort and reduce. In the copy step the Reducers fetch the output maps from the tasktracker and store them on disk or in memory. This is where the mapreduce.tasktracker.http.threads, which has a default setting of 40, and the mapred.reduce.parallel.copies parameters come in to play. The first parameter is the number of threads that the tasktracker uses to track the different data nodes and partition data in the cluster. Increase this value on larger clusters. The other is the number of parallel copies that take place during the copy and shuffle steps. The mapred.reduce.parallel.copies parameter has a default value of 5 and should also be increased, but the amount of increase depends on the hardware on which the Reducers and tasktrackers are running.

The Mapping process output data is copied into memory, and the amount of this memory is controlled by two parameters — the mapred.job.shuffle.input.buffer.percent(0.70) and the mapred.child.java.opts(-Xmx200). Increasing the mapred.child.java.opt to 1G, or higher if the memory is available, usually keeps whole copy, shuffle and merge steps in memory, which decreases overall execution time and increases MapReduce performance. If the copy, shuffle and merge steps still spill to disk, check the mapred.job.shuffle.merge.percent(0.66), mapred.inmem.merge.threshold(1000) and the io.sort.factor(10) parameters. The mapred.job.shuffle.merge.percent parameter determines the percentage of the memory buffer that the copy, shuffle merge steps are allowed to fill before spilling to disk. Increasing this percentage can help keep the merge process in memory.

By default once there are 1000 Mapping process outputs in memory the outputs start to spill to disk. To stop this from happening, set the mapred.inmem.merge.threhold to 0 and allow memory to fill with Mapping process outputs until the mapred.job.shuffle.merge.percent value is reached.

The last parameter in this process is the io.sort.factor, which determines the number of Mapping process output streams that can be merged at once. Depending on how fast the Reducer is processing the merged files, increasing this parameter can get the merged files to the Reducer faster. At this point the all the merged files are normally sent to a disk buffer for the Reducer to work on them. Changing the following mapred.job.reduce.input.buffer.percent(0.0) parameter to a value from 0.60 to 0.80 keeps the merged files in memory and out of the Reducer disk buffer, decreasing Reducer execution time. This memory buffer comes from the overall JVM memory heap size. By
experimenting and tuning the `job.shuffle.input.buffer.percent` and the `mapred.job.reduce.input.buffer.percent`, it is possible to keep the whole MapReduce process in memory until the Reducer creates the output file. By tuning the parameters in the MapReduce section the benchmarking Hadoop cluster created for this white paper had a 12 percent decrease in normalized execution time when running a Teragen / Terasort job of 1TB.

![Map output fetching work flow](image)

**Java reuse option**

Hadoop supports a configuration parameter called `mapred.job.reuse.jvm.num.tasks` that governs whether Map/Reduce JVM processes spawned are reused for running more than one task. This property can be found in the `mapred-site.xml` file. 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. The JVM spends less time interpreting Java byte code, because some of the earlier tasks are expected to trigger JIT compilation of hot methods. JVM reuse is expected to specifically benefit scenarios in which there are a large number of very short running tasks. We noticed close to five percent improvement in performance by enabling JVM reuse.

**Summary**

By changing all of the parameters discussed in the Hadoop framework there can be up to a 38 percent decrease in normalized execution time. Changing these settings also results in other benefits, such as decreased space used on the HDFS partitions, allowing for bigger Hadoop workloads, due to the use of compression.
Conclusion

This white paper has explained the need for addressing certain issues in the Linux OS — which is the foundation of Hadoop — to help with job completion; and it has explained the need to consider hardware choices before creating the Hadoop cluster. Understanding the reasons why certain system requirements must be met is key to achieving the best performance in your Hadoop cluster. These hardware choices affect the ability to expose more tuning options within the Hadoop parameters. In this white paper we have tried to explain the reasons why Hadoop settings must be changed from their default settings and how those changes can help overall Hadoop performance. It is now up to you to explore these parameters to better tune you Hadoop cluster.
Appendix A — LZO installation process

The installation consists of the following steps:

- Installing LZO
  
  `sudo yum install lzop` or `apt-get install lzop`
  
  `sudo yum install lzo-devel` or `apt-get install lzo-devel`

- Installing ANT
  
  `sudo yum install ant ant-nodeps ant-junit java-devel`
  
  `apt-get install ant ant-nodeps ant-junit java-devel`

- Downloading the source
  
  `git clone https://github.com/twitter/hadoop-lzo`

- Compiling Hadoop-LZO
  
  `ant compile-native tar`

For further instructions and troubleshooting see [https://github.com/twitter/hadoop-lzo](https://github.com/twitter/hadoop-lzo)

- Copying Hadoop-LZO jar to Hadoop jars
  
  `sudo cp build/hadoop-lzo*.jar /usr/lib/hadoop/lib/`

- Moving native code to Hadoop native jars
  
  `sudo mv build/hadoop-lzo-0.4.17-SNAPSHOT/lib/native/Linux-amd64-64/ /usr/lib/hadoop/lib/native/`
  
  `cp /usr/lib/hadoop/lib/native/Linux-amd64-64/libgplcompression.* /usr/lib/hadoop/lib/native/`

  Correct version number with the version you cloned

- When working with a real cluster (as opposed to a pseudo-cluster) you need to resynchronize these to the rest of the machines
  
  `rsync /usr/lib/hadoop/lib/` to all hosts.

You can dry run this first with `-n`

- Login to Cloudera Manager
- Select from Services: mapreduce1->Configuration
- Client->Compression
- Add to Compression Codecs:
com.hadoop.compression.lzo.LzoCodec
com.hadoop.compression.lzo.LzopCodec

- Search "valve"
- Add to MapReduce Service Configuration Safety Valve
  
  ```
  io.compression.codec.lzo.class=com.hadoop.compression.lzo.LzoCodec
  mapred.child.env="JAVA_LIBRARY_PATH=/usr/lib/hadoop/lib/native/Linux-amd64-64/"
  ```

- Add to MapReduce Service Environment Safety Valve
  
  ```
  HADOOP_CLASSPATH=/usr/lib/hadoop/lib/*
  ```
Appendix B — Hadoop cluster hardware configuration

Table 3. Dell C8220 DN cluster hardware configuration

<table>
<thead>
<tr>
<th>Hardware Details</th>
<th>Dell C8220 DN Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of nodes</td>
<td>4</td>
</tr>
<tr>
<td>Processor, per node</td>
<td>2X E5 2640(2.5GHz, 55W)</td>
</tr>
<tr>
<td>Memory, per node</td>
<td>8 x 8GB 1333MHz RDIMMs</td>
</tr>
<tr>
<td>HDD, per node</td>
<td>12 x 1TB 2.5&quot; 7.2k SATA</td>
</tr>
<tr>
<td>HBA</td>
<td>LSI 2008</td>
</tr>
<tr>
<td>BMC firmware revision</td>
<td>1.28</td>
</tr>
<tr>
<td>Power Supply</td>
<td>4x 1400W</td>
</tr>
<tr>
<td>BIOS Options</td>
<td>2.03</td>
</tr>
<tr>
<td>BIOS revision</td>
<td>1.20</td>
</tr>
<tr>
<td>FBC revision</td>
<td>Enabled</td>
</tr>
<tr>
<td>IOMMU</td>
<td>Enabled</td>
</tr>
<tr>
<td>HW Prefetch</td>
<td>Disabled</td>
</tr>
<tr>
<td>HW Prefetch Training on SW</td>
<td>Enabled</td>
</tr>
<tr>
<td>SVMM</td>
<td>Enabled</td>
</tr>
<tr>
<td>SR-IOV</td>
<td>Disabled</td>
</tr>
<tr>
<td>Turbo</td>
<td>Auto(1600)</td>
</tr>
<tr>
<td>Memory Speed</td>
<td>Disabled</td>
</tr>
<tr>
<td>PM MEZZ.LOM, PCI-e</td>
<td>N/A</td>
</tr>
<tr>
<td>ALL nonused USB ports disabled</td>
<td>Disabled</td>
</tr>
<tr>
<td>RM disabled / RM LOM Shared</td>
<td>Enabled</td>
</tr>
<tr>
<td>SATA Power Management</td>
<td>Max Performance</td>
</tr>
<tr>
<td>Power Mgmt Mode</td>
<td>N/A</td>
</tr>
<tr>
<td>OS config</td>
<td>JVM version</td>
</tr>
<tr>
<td></td>
<td>OS version</td>
</tr>
<tr>
<td></td>
<td>1.6.31</td>
</tr>
<tr>
<td></td>
<td>CentOS 6.4</td>
</tr>
</tbody>
</table>
### Table 4. Dell R620 M,S,E,HA

<table>
<thead>
<tr>
<th>Hardware Details</th>
<th>Dell R620 M,S,E,HA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of nodes</td>
<td>3</td>
</tr>
<tr>
<td>Processor, per node</td>
<td>E5-2680 (2.7GHz, 130W)</td>
</tr>
<tr>
<td>Memory, per node</td>
<td>8 x 8GB 1600MHz RDIMMs</td>
</tr>
<tr>
<td>HDD, per node</td>
<td>8 x 146GB 2.5&quot; 15k SAS</td>
</tr>
<tr>
<td>HBA</td>
<td>H310</td>
</tr>
<tr>
<td>BMC firmware revision</td>
<td>1.11</td>
</tr>
<tr>
<td>Power Supply</td>
<td>2x 1100W</td>
</tr>
<tr>
<td>BIOS Options</td>
<td>3.0.0</td>
</tr>
<tr>
<td>BIOS revision</td>
<td>1.20</td>
</tr>
<tr>
<td>FBC revision</td>
<td>Disabled</td>
</tr>
<tr>
<td>IOMMU</td>
<td>Enabled</td>
</tr>
<tr>
<td>HW Prefetch</td>
<td>Enabled</td>
</tr>
<tr>
<td>HW Prefetch Training on SW</td>
<td>Enabled</td>
</tr>
<tr>
<td>SVMM</td>
<td>Disabled</td>
</tr>
<tr>
<td>SR-IOV</td>
<td>Auto</td>
</tr>
<tr>
<td>Turbo</td>
<td>Auto(1333)</td>
</tr>
<tr>
<td>Memory Speed</td>
<td>Disabled</td>
</tr>
<tr>
<td>PM MEZZ.LOM, PCI-e</td>
<td>N/A</td>
</tr>
<tr>
<td>ALL nonused USB ports disabled</td>
<td>N/A</td>
</tr>
<tr>
<td>RM disabled / RM LOM Shared</td>
<td>Disabled</td>
</tr>
<tr>
<td>SATA Power Management</td>
<td>Disabled</td>
</tr>
<tr>
<td>Power Mgmt Mode</td>
<td>Max Performance</td>
</tr>
<tr>
<td>OS config</td>
<td>N/A</td>
</tr>
<tr>
<td>JVM version</td>
<td>1.6.31</td>
</tr>
<tr>
<td>OS version</td>
<td>CentOs 6.4</td>
</tr>
</tbody>
</table>
Appendix C — Glossary of Hadoop parameters affecting performance

dfs.namenode.handler.count

The number of threads the NameNode uses to serve requests. The default value is 10, changing the value gained no noticeable change in performance in the test cluster. With larger clusters changing this value may increase performance, because there are more file operations on the name node. Change the value, and then use a tool such as nnbench for verification.

dfs.datanode.handler.count

This value is the number of threads that the data nodes use. The default value is 3, changing this value to the number of hard drives in the data node seems to yield the best results. Use testDFSio to verify if this works for your cluster.

dfs.datanode.max.xcievers

This value is the maximum number of threads used to access the local file system on a data node. The default value is 256. Increasing this to 2048 increased performance on the test cluster. If the data nodes in your cluster have more than 12 hard drives attached, increasing this value even more might increase data node HDFS performance.

io.file.buffer.size

This is memory buffer to which file IO copies, stores and writes data. It should be a multiple of 4096. It should be safe to use 131072, but we used double that value. The performance gain is not enormous. When using HBase be careful not to set this value too high.

dfs.block.size (hdfs-site.xml)

This parameter controls the block size written by the data node on to the HDFS file system. A larger the value in megabytes is generally better, until bottlenecks such as the storage controller or Memory come into play.

io.sort.factor

This parameter is the number of streams to merge concurrently when shuffling and sorting files.

io.sort.mb

This parameter is the amount of memory used to buffer Mapping process output sorting.

mapred.reduce.parallel.copies

This parameter is the number of concurrent connections that the Reducer uses to fetch data from the Mappers.

tasktracker.http.threads

This parameter is the number of connections that the tasktracker uses to provider intermediate data to the Reducers.
mapred.tasktracker.map.tasks.maximum
mapred.tasktracker.reduce.tasks.maximum

These parameters control the maximum number of Mapping and Reducer tasks that run on a data node. A good general rule is to use one-half to two times the number of cores on a node.

mapred.max.split.size
mapred.min.split.size

These two parameters, along with the dfs.block.size, determine the data-chunk size that is fed into the MapReduce process.

mapred.map.tasks.speculative.execution
mapred.reduce.tasks.speculative.execution

These parameters cause the jobtracker to run a copy of the same job on another node. Once the data output is done on one of the data nodes, the other incomplete job is ended.

mapred.job.reuse.jvm.num.tasks

This parameter tells Hadoop to use the JVM already created for a task, instead destroying and recreating the JVM. When set to −1 this setting can give up to a five percent performance increase in MapReduce jobs.

mapred.compress.map.output
mapred.map.output.compression.codec
mapred.output.compress
mapred.output.compression.type

These parameters tell Hadoop what compression codec and type to use, as well as in what phase of the MapReduce process to use it. Compression helps with faster disk writes, saves HDFS partition space and decreases the IO transfer time between the Mapper and the Reducers. These benefits come, however, at the cost of processor cycles to encode and decode the compression stream.

MapReduce parameters that affect timing

The following parameters can help reduce the latencies inherent in the execution process of MapReduce. These parameters can be added or found in the mapred-site.xml configuration file.

mapreduce.tasktracker.outofband.heartbeat

Setting the mapreduce.tasktracker.outofband.heartbeat parameter to true, instead of using the default of false, allows the tasktracker to send an out-of-band heartbeat when the task is completed, to reduce latency.

jobclient.progress.monitor.poll.interval

By default the jobclient.progress.monitor.poll.interval is set 1000 milliseconds. Setting this parameter lower on small clusters can decrease the time lost waiting to verify job completion, because this parameter reports the status of the Hadoop job while it is running.
mapreduce.jobtracker.heartbeat.interval.min

This is the interval at which each service checks on each other when a job is running. On small clusters changing the `mapreduce.jobtracker.heartbeat.interval.min` from the default of 10 to a smaller value can increase performance.

mapred.reduce.slowstart.completed.maps

This parameter tells the Reducer phase of MapReduce to start immediately when the Mapping phase starts, instead of waiting for Mapping process output files to be created. Having the Reducer process spun up and ready to go can increase performance of the whole MapReduce process. Set the `mapred.reduce.slowstart.completed.maps` to 0 on small jobs; larger job must be set higher.
## Appendix D — Optimal Hadoop parameters for test cluster

Table 5. Optimal Hadoop parameters for test cluster

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>core-site.xml</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>io.file.buffer.size</td>
<td>131072</td>
<td>Size of read/write buffer used in sequence files.</td>
</tr>
<tr>
<td><strong>hdfs-site.xml</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dfs.blocksize</td>
<td>404750336</td>
<td>HDFS blocksize 384MB for large file systems.</td>
</tr>
<tr>
<td>dfs.namenode.handler.count</td>
<td>100</td>
<td>NameNode server thread to handle RPCs.</td>
</tr>
<tr>
<td><strong>mapred-site.xml</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mapreduce.output.compress</td>
<td>true</td>
<td>compress map output</td>
</tr>
<tr>
<td>mapreduce.output.compression.type</td>
<td>record</td>
<td>The file compression type.</td>
</tr>
<tr>
<td>mapreduce.output.compression.codec</td>
<td>lzo</td>
<td>sets the compression codec to be used.</td>
</tr>
<tr>
<td>mapreduce.map.java.opts</td>
<td>Xmx1536M</td>
<td>Larger memory heap-size for child jvms, during Map process.</td>
</tr>
<tr>
<td>mapreduce.task.io.sort.mb</td>
<td>450</td>
<td>Higher memory limit while sorting data for map tasks.</td>
</tr>
<tr>
<td>mapreduce.task.io.sort.factor</td>
<td>100</td>
<td>More streams merged at once while sorting files.</td>
</tr>
<tr>
<td>mapreduce.job.shuffle.merge.percent</td>
<td>0.66</td>
<td>memory reserved for storing map outputs.</td>
</tr>
<tr>
<td>mapred.job.reduce.input.buffer.percent</td>
<td>0.65</td>
<td>% of jvm memory used to merge map output files</td>
</tr>
<tr>
<td>mapred.job.reuse.jvm.num.tasks</td>
<td>-1</td>
<td>Reuses JAVA JVM over and over</td>
</tr>
<tr>
<td>mapred.reduce.slowstart.completed.maps</td>
<td>0</td>
<td>Starts the reducer before the mapper is finished</td>
</tr>
<tr>
<td>mapreduce.reduce.shuffle.parallelcopies</td>
<td>50</td>
<td>Higher number of parallel copies run by reduces to fetch outputs from very large number of maps.</td>
</tr>
</tbody>
</table>
Appendix E — Further information

For information on the Dell Crowbar tool for installation of the Hadoop cluster, see Dell Cloudera Apache Hadoop install with Crowbar.

For information on the HiBench test suite for Hadoop clusters, see GitHub.com/HiBench.

For more information on using HiBench, see the Dell white paper, Dell Apache Hadoop Performance Analysis.

See the Dell TechCenter blog, Getting the Right Mix for Hadoop.