AMD

Hadoop Tuning Guide for AMD EPYC™ Processor Based Servers

Publication # **56419** Revision: **1.0**
Issue Date: **October 2018**

Advanced Micro Devices

Contents

Revision History

Date	Revision	Description
October 2018	1.00	Initial NDA release.

Chapter 1 Introduction

The Apache™ Hadoop® software library is a framework that allows for distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering either local computation and storage or both. When implemented in a cluster, the software has the built-in resiliency to handle everything from a failed drive to an entirely failed server. Hadoop uses these techniques instead of relying on hardware to delivery high-availability.

With such a wide diversity of applications and environments in which Hadoop runs there is not a golden rule for tuning the cluster. Different Hadoop distribution vendors expose different settings through their management software.

This guide will help provide suggestions as to which parameters will have the most impact on the performance of Hadoop clusters. The categories of parameters discussed in this paper include Basic Input Output System (BIOS), Linux™ OS, Hadoop Distributed File System (HDFS), and Yet Another Resource Manager (YARN)

The concepts disclosed in this document while tested on Cloudera CDH 5.15 should apply to Hadoop distributions based upon Hadoop 2.x, including Cloudera, MapR, and Hortonworks.

1.1 Challenges involved in tuning Hadoop

Hadoop is a sophisticated software framework which comprises several components interacting with each other across multiple systems. Bottlenecks in either of these components can and will cause poor performance of the Hadoop workload. Therefore, Hadoop performance is affected by every element of the system including BIOS settings, the operating system, network and disk I/O, and Hadoop stack configuration. Hadoop has several configuration settings that can impact performance thus necessitating a certain degree of familiarity with the inner workings of the Hadoop ecosystem to optimize the configuration settings. As each changes to the settings can influence others, Hadoop tuning becomes an iterative process in which the implications of a particular change in settings needs to be well understood before moving onto other changes in tuning.

Chapter 2 BIOS Settings

One of the first places to start for tuning systems for running Hadoop is in the BIOS. The parameter names and options may vary slightly between the OEM's. The following bios settings, using the names from the HPE DL385 Gen10 systems, have the most significant impact on performance of Hadoop systems. Please note that all of these settings allow for an increased raw performance value but may result in lower power efficiency.

1. **PerformanceDeterminism: PerformanceDeterministic**. Allows the system to use more power to maintain peak performance. Please see document "*https://www.amd.com/system/files/2017-06/Power-Performance-Determinism.pdf"* for more information on AMD EPYC's Determinism slider.
2. **ProcSMT: Enabled**. Enables Symmetric Multi-Threading (SMT), which creates two computing threads per core.
3. **ProcAMDBoost: Enabled**. Allows the CPU cores to boost to higher frequencies when circumstances allow for it.
4. **ThermalConfig: MaxCooling**. Runs the fans at maximum speed to keep the system cooler, which allows more frequent and longer lasting boosts in frequencies of the cores.
5. **MinProcIdlePower: C6**. Determines whether the processor cores can go into idle states when not being used. Depending upon your workload characteristics this could be set to either of these. It is better to set this to C6 here and disable any CStates that you don't want at the O/S level, since it is something that can be changed without requiring a restart of the system.
6. **AmdMemoryInterleaving=Auto**. Controls the memory interleaving mode. It can be set to channel, die, socket, or Auto. HPE's auto setting will select the highest performing mode based on the memory population. If memory is correctly populated, channel interleaving will be utilized.

Chapter 3 Linux optimizations

Tuning the Linux operating system for Hadoop involves looking at the interaction of the various subcomponents of a system particularly, the network and disk I/O, which often conflict with each other. The goal of this tuning is to get each subsystem optimized for throughput and address any performance issues or bottlenecks with simple stress test tools. This will rule out any performance issues in the subsystems before running with tuning the Hadoop instance.

3.1 Network Configuration

It is recommended to read "*Linux® Network Tuning Guide for AMD EPYC™ Processor Based Systems*" which discusses how to properly configuring the network for systems in a Hadoop cluster. Following the principles outlined in the paper: make sure to tune the size of the TX and RX rings; change the number of interrupts queues to match the cores on the NUMA node which the NIC is collocated; and pin those interrupts to the correct cpu cores. The iperf utility can be used to stress test the network infrastructure to ensure that it is setup properly. Please follow the tuning guide specifically in the setting for the iommu for the O/S, as this will have significant impact on system performance. This is normally done by setting the iommu to pass-through mode by adding the kernel parameter, `iommu=pt`, on the kernel boot line (for RHEL 7.5 this is done by modifying the `/etc/default/grub` file and running grub2-mkconfig utility).

3.2 Disk Configuration

Disk I/O: To optimize the disk I/O in a Hadoop environment, the recommended things to look at are: partition alignment, scheduler, queue_depth, number of requests allowed, and filesystem mount options. The alignment of partitions is more critical when using SSD and NVMe drives, otherwise, a significant performance impact when misaligned will increase number of I/O operations going to the device.

The scheduler option of choice for most Hadoop environments will be either deadline or noop. While there is no standard on which to use, it is recommended that you test them both in the environment. Changing the scheduler can be done systemwide on the linux boot command line with the kernel option of `elevator=<scheduler>`. The scheduler for a device can be viewed and modified after boot sequence via the file `/sys/block/<device>/queue/scheduler`. For example, the square brackets "[]" are around deadline indicating that this device is configured to use the deadline scheduler. Then we echo `noop` into the file for device sda and then list the contents of the file again to see that the square brackets are now around the noop scheduler.

```
cat /sys/block/sda/queue/scheduler
noop [deadline] cfq
echo noop > /sys/block/sda/queue/scheduler
cat /sys/block/sda/queue/scheduler
[noop] deadline cfq
```

Queue_depth and number of requests (nr_requests) should be set such that the device is not being overwhelmed. These nr_requests file is in the `/sys/block/<device>/queue` directory. While the queue_depth file is in the `/sys/block/<device>/device` directory. This is indicated by large queue times as seen by performance monitoring utilities such as sar or iostat

Most Hadoop deployments, with the exception of MapR™ (which deploys on unformatted disks) on Linux today, are using either ext4 or xfs filesystems. The important filesystem options always include noatime for any Hadoop data mounts. If running on an SSD, and it is supported, include the discard option. These options can be set with the -o parameter when manually executing the mount command, for example `mount -o noatime /dev/sda1 /data/1`, or by modifying the `/etc/fstab` file for filesystems that are mounted during the boot process. An example `/etc/fstab` file could look like this:

```
UUID=a6f3ee72-5fef-4ec5-8074-49f4ae109a88 /data/1  xfs defaults,noatime 0 0
UUID=30375f23-b704-4b3b-ae33-349bb0dac416 /data/2  xfs defaults,noatime 0 0
UUID=0c347985-6c1f-469e-918e-ae6ad4ee5002 /data/3  xfs defaults,noatime 0 0
UUID=a970c80a-c9ea-47c4-9fbe-acddf9ab7ffb /data/4  xfs defaults,noatime 0 0
UUID=69045795-a472-477c-9115-35bc4b940224 /data/5  xfs defaults,noatime 0 0
UUID=3531c522-3f82-4b9e-876f-ee44e133dc3c /data/6  xfs defaults,noatime 0 0
UUID=fa96cd17-37a0-441b-b001-527e7765de42 /data/7  xfs defaults,noatime 0 0
UUID=5036062a-06a2-4227-98b3-a1224cb3f649 /data/8  xfs defaults,noatime 0 0
```

The fio test can be used to stress test the disk subsystems. Use this to determine maximum disk throughput in both I/O per second (iops) and MB/sec. This utility can also see if any disks are running slower than the rest.

The default number of open files is small and will likely cause `java.io.FileNotFoundException` (Too many open files) to occur and jobs will fail. To avoid this, increase the number of open files to something like 32768 or even 65536 using ulimit command as root user, `ulimit -n 65536` or adding the following line to the `/etc/security/limits.conf` file.

```
*       -   nofile   65536
*       -   nproc 65536
```

The system wide setting of fs.file-max may need to increase in some cases. This can be done on the fly through the sysctl command, `sysctl -w fs.file-max <maximum number of open files>` or made persistent across reboots by adding `fs.file-max=<maximum number of open files>` in `/etc/sysctl.conf` file.

3.3 Memory Configuration

It is important that the memory is populated in a way that all memory channels are being used. Place the same number and size memory DIMMs in each memory channel as this will keep the NUMA nodes balanced. Minimize swapping with O/S sysctl utility of vm.overcommit_memory and vm.swappiness. Tune down the O/S cache buffers with vm.dirty_ratio and vm.dirty_background_ratio. All four of these parameters can be modified while at runtime of a system by using the sysctl command, for example, `sysctl -w vm.overcommit_memory=0 vm.swappiness=1 vm.dirty_ratio=20 vm.dirty_background_ratio=1`, or by adding them to `/etc/sysctl.conf` file to keep the changes persistent across reboots.

3.4 CPU Configuration

Check that the performance governor on all cores. On RHEL 7, view the frequency governors available using `cpupower frequency-info –governor` command. To set the frequency governor use `cpupower frequency-set –governor performance`. Generally, the tuned-adm throughput-performance profile works best for Hadoop workloads, and this will set the governor to performance.

3.5 Example configuration files RHEL 7.5 Server

```
/etc/default/grub
GRUB_CMDLINE_LINUX="crashkernel=auto rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap
rhgb quiet iommu=pt"

/etc/rc.local
#configure Mellanox network card.
mlnx_tune -p HIGH_THROUGHPUT
set_irq_affinity_bynode.sh 3 ens2f0 ens2f1
#disable transparent hugepages
echo never > /sys/kernel/mm/transparent_hugepage/defrag
echo never > /sys/kernel/mm/transparent_hugepage/enabled
#disk tuning
for a in sda sdb sdc sdd sde sdf sdg sdh
do
  echo “deadline” > /sys/block/${a}/queue/scheduler
  echo 512 > /sys/block/${a}/queue/nr_requests
done

/etc/sysctl.conf
#Network Tunings
net.ipv4.conf.default.rp_filter=1
net.ipv4.tcp_timestamps=0
net.ipv4.tcp_sack = 1
net.core.netdev_max_backlog = 25000
net.core.rmem_max = 2147483647
net.core.wmem_max = 2147483647
net.core.rmem_default = 33554431
net.core.wmem_default = 33554432
net.core.optmem_max = 33554432
net.ipv4.tcp_rmem =8192 33554432 2147483647
net.ipv4.tcp_wmem =8192 33554432 2147483647
net.ipv4.tcp_low_latency=1
net.ipv4.tcp_adv_win_scale=1
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv4.conf.all.arp_filter=1
net.ipv4.tcp_retries2=5
net.ipv6.conf.lo.disable_ipv6 = 1
net.core.somaxconn =  65535
#memory cache settings
vm.swappiness=1
vm.overcommit_memory=0
vm.dirty_background_ratio=1
```

Chapter 4 Hadoop Settings

The Hadoop Distributed File System (HDFS) and Yet Another Resource Negotiator (YARN) are the main components that form the data management layer for Hadoop. HDFS is the storage management framework for Hadoop while Yarn is the resource management framework for Hadoop. There is no universal set of settings that will work for every Hadoop environment, and they will be highly dependent upon workload characteristics and hardware subsystem choices. However, the following sections will give a brief description of the various interactive components of Hadoop, and some guidelines on where to start with the tuning to come up with baselines.

4.1 YARN Settings

Within YARN there are two major component daemons. These are the ResourceManager and the NodeManager. The ResourceManager is the master daemon of YARN and it manages the assignment of resources among the competing applications. The most significant part of the ResourceManager is the scheduler as most Hadoop installations will rely on either the Fair scheduler, which tries to assign resources equally across jobs over time, or the Capacity scheduler, which tries to give each user or job a minimum capacity guarantee, while allowing any excess capacity not utilized by one user or organization to be accessed by others. If only one job is running at a time, then either scheduler will work.

The main properties for proper YARN configuration will fall into the Resource allocation minimums and maximums. The various Hadoop vendors have worksheets and scripts that will recommend settings for some of these based upon what other Hadoop components (HIVE, Spark, etc.) you have installed on your nodes. These are the main settings for resource consumption controls by the scheduler. They define the minimum, maximum, and stepping allocation of memory and vcore (virtual CPU) resources of the nodes in the cluster. In many cases, YARN, the MapReduce concept will be breaking a job up into hundreds or even thousands of tasks, and reduce the task footprint to a single thread and smaller memory footprint. This will represent the minimum container size allocations for both memory and vcores. When calculating these values most of the time it's a matter of dividing total memory available to the YARN NodeManager (yarn.nodemanager.resource.memory-mb) by number of vcores (yarn.nodemanager.resource.vcores) available to it. In other cases, there will be large container applications (i.e. SPARK or Hive with LLAP) where the container's memory footprint and number of cores can get fairly large. These latter cases are going to involve much more consideration of the NUMA architecture of the AMD EPYC processor. Namely, the number of CPU cores, and the amount of memory available per NUMA node. These settings will be set either in the management software of the cluster (ClouderaManager for CDH, Ambari for HDP, MapR Control System for MapR) or by modifying the yarn-site.xml files directly in a manual Apache Hadoop configuration.

`yarn.scheduler.minimum-allocation-vcores` (set to lowest number of vcores per container normally 1)
`yarn.scheduler.maximum-allocation-vcores` (set to maximum number of vcores per container, normally restrict this to the number of vcores in a NUMA node or less)
`yarn.scheduler.increment-allocation-vcores` (normally 1, incremental step from min to max will be multiple of this number)
`yarn.scheduler.minimum-allocation-mb` (default 1024, but should be <=yarn.nodemangager.resource.memory-mb/yarn.nodemanager.resource-cpu-vcpus in order to utilize as many cores as possible and effective use the memory).
`yarn.scheduler.maximum-allocation-mb` (maximum desired container size, which will be application dependent, but recommend keeping it less than the memory per NUMA node in system)
`yarn.scheduler.increment-allocation-mb` (incremental steps from min-allocation to max-allocation will be multiples of this number. 1024 would mean 1 GB increments.)
`yarn.nodemanager.resource.memory-mb` (set to Total memory in the worker node to be used by nodemanager i.e. system memory less any reserved space for O/S overhead.)
`yarn.nodemanager.resource.cpu-vcores` (set to total number of virtual cores in the worker node to be used by nodemanager)

So for example with a datanode that has dual EPYC 7451 (24 core/48 thread per socket) with 512 GB of memory, you could would configure as so:

```
yarn.scheduler.minimum-allocation-vcores: 1
yarn.scheduler.maximum-allocation-vcores: 6
yarn.scheduler.increment-allocation-vcores: 1
yarn.scheduler.minimum-allocation-mb: 1024
yarn.scheduler.maximum-allocation-mb: 63488
yarn.scheduler.increment-allocation-mb: 1024
yarn.nodemanager.resource.memory-mb: 507904
yarn.nodemanager.resource.cpu-vcores: 96
yarn.nodemanager.local-dirs: a list of local directories (one per disk) that
yarn will use to store intermediate and temporary data.
```

Using Native task libraries on map outputs can be a significant boost in performance. The native libraries include components for compression codecs (bzip2, lz4, snappy, and zlib), native I/O utilities for HDFS Short-Circuit Local Reads and Centralized Cache Management in HDFS, and CRC32 checksum implementation. In high I/O intensive applications, similar to the Hadoop dfsio test using these libraries can significantly improve your performance. In the latest CDH platforms (5.14) there is a simple checkbox option called “Enable Optimized Map-side Output Collector” which allows this to be enabled.

To verify that the native libraries are properly installed run the following command and check the output:

```
hadoop checknative -a
18/08/23 15:11:32 INFO bzip2.Bzip2Factory: Successfully loaded & initialized
native-bzip2 library system-native
18/08/23 15:11:32 INFO zlib.ZlibFactory: Successfully loaded & initialized
native-zlib library
Native library checking:
```

```
hadoop:  true /opt/cloudera/parcels/CDH-5.15.0-
1.cdh5.15.0.p0.21/lib/hadoop/lib/native/libhadoop.so.1.0.0
zlib:    true /lib64/libz.so.1
snappy:  true /opt/cloudera/parcels/CDH-5.15.0-
1.cdh5.15.0.p0.21/lib/hadoop/lib/native/libsnappy.so.1
lz4:     true revision:10301
bzip2:   true /lib64/libbz2.so.1
openssl: true /lib64/libcrypto.so
```

It is also recommended to check the map task system logs where expected native task utilization is enabled and look for the following message(s):

```
2018-08-23 15:30:36,019 INFO [main]
org.apache.hadoop.mapred.nativetask.NativeRuntime: Nativetask JNI library
loaded.
2018-08-23 15:30:36,053 INFO [main]
org.apache.hadoop.mapred.nativetask.NativeBatchProcessor: NativeHandler:
direct buffer size: 1048576
2018-08-23 15:30:36,075 INFO [main]
org.apache.hadoop.mapred.nativetask.util.OutputUtil:
nativetask.output.manager =
org.apache.hadoop.mapred.nativetask.util.NativeTaskOutputFiles
2018-08-23 15:30:36,082 INFO [main]
org.apache.hadoop.mapred.nativetask.NativeMapOutputCollectorDelegator:
Native output collector can be successfully enabled!
2018-08-23 15:30:36,082 INFO [main] org.apache.hadoop.mapred.MapTask: Map
output collector class =
org.apache.hadoop.mapred.nativetask.NativeMapOutputCollectorDelegator
```

4.2 HDFS Settings

On the HDFS side of things there is little to impact overall performance. The main parameters include the following, which will need to be tested with the workload. These settings will be set either in the management software of the cluster (ClouderaManager for CDH or Ambari for HDP) or by modifying the hdfs-site.xml files directly. MapR has their own filesystem which is managed through the MapR Control System.

```
NameNode java heap size
DataNode java heap size
dfs.datanode.data.dir: list of directories in which HDFS data will be stored
(one per disk).
dfs.blocksize:  default size of hdfs blocks.  Can be overridden by the
requested job.
dfs.namenode.handler.count: number of server threads for the NameNode.
Dfs.datanode.handler.count: number of server threads for the DataNode.
```