Performance Tuning HBase and Hadoop.

Using HBase in production often requires that you turn many knobs to make it hum as expected. More Here `http://hbase.apache.org/0.94/book/performance.html`

Server Hardware Details.

Based on these parameter we will be setting the configuration below.

```
RAM          : 48GB
CPUs         : 40 Cores
Ethernet     : 1G
```

Hadoop YARN Architecture.

Courtesy : Hadoop

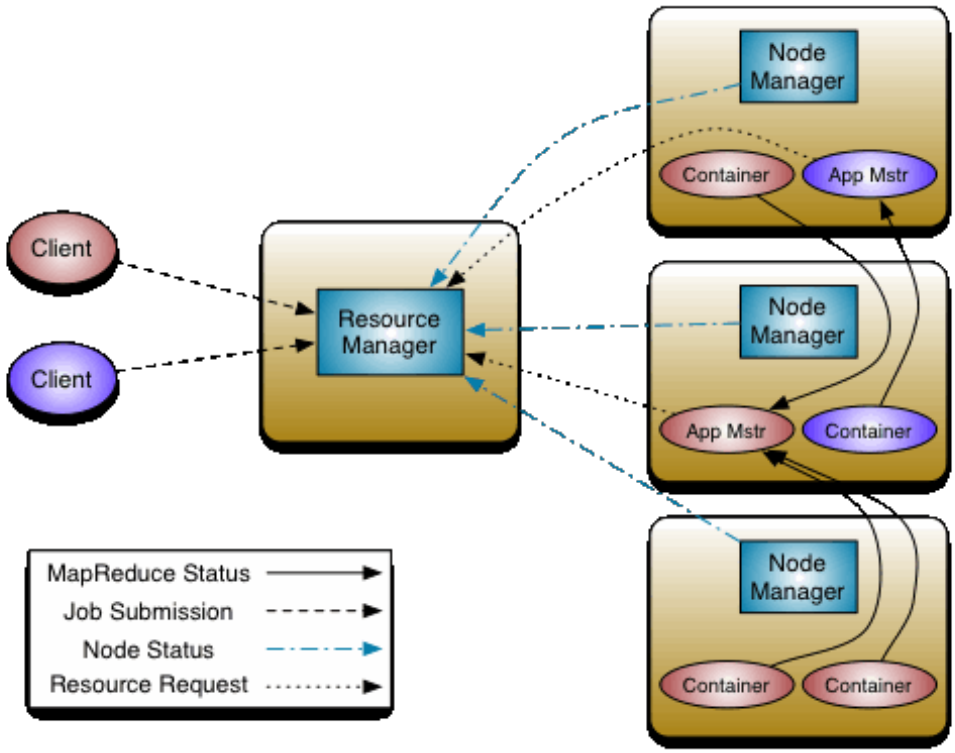


Figure 1: alt text

Step By Step - YARN Workflow.

1. Run/Submit `job` from the client node.
2. Get new application an submit to `resourcemanager`.
3. Copy `job` resources from shared filesystem such as HDFS.
4. Submit application to `resourcemanager`
5. Submit to `nodemanager`
    - a. Start Container
    - b. Lunch `MRAppMaster`
6. Initialize `job`.
7. Retrieve input splits.
8. `MRAppMaster` will send `allocate resourcces` to `resourcemanager` to inform about the current resources used.
9. `MRAppMaster` sends required to specific `nodemanager`.
    - a. Start Container
    - b. Launch `task JVM`.
10. `Child` retrieves job resources from shared filesystem (HDFS).
11. `run` MapTask or Reduce Task.

Diagram

HBASE `JAVA_OPTS` configuration.

Cloudera on Hbase Memory : "RAM, RAM, RAM. Don't starve HBase.". So the primary goal is to not starve `hbase` of any RAM. If you have RAM then feed `hbase`.

Cloudera on Swapping : "Watch out for swapping. Set swappiness to 0".

Details of Options used in `hbase`.

1. Use `-XX:+ParallelRefProcEnabled` When this flag is turned on, GC uses multiple threads to process the increasing references during Young and mixed GC. With this flag for HBase, the GC remarking time is reduced by 75%, and overall GC pause time is reduced by 30%.

2. Set `-XX:-ResizePLAB` and `-XX:ParallelGCThreads`=8+(logical processors-8)(5/8) Promotion Local Allocation Buffers (PLABs) are used during Young collection. Multiple threads are used. Each thread may need to allocate space for objects being copied either in Survivor or Old space. PLABs are required to avoid competition of threads for shared data structures that manage free memory. Each GC thread has one PLAB for Survival space and one for Old space. We would like to stop resizing PLABs to avoid the large communication cost among GC threads, as well as variations during each GC.

```
-XX:ParallelGCThreads=8+(logical processors-8)(5/8)
-XX:ParallelGCThreads= 8+(40-8)(5/8)=28
```

The following are the recommended Java GC and HBase heap settings:

Step 1. Give HBase enough heap size by editing the `hbase-env.sh` file. For example, the following snippet configures a 16GB heap size for HBase:

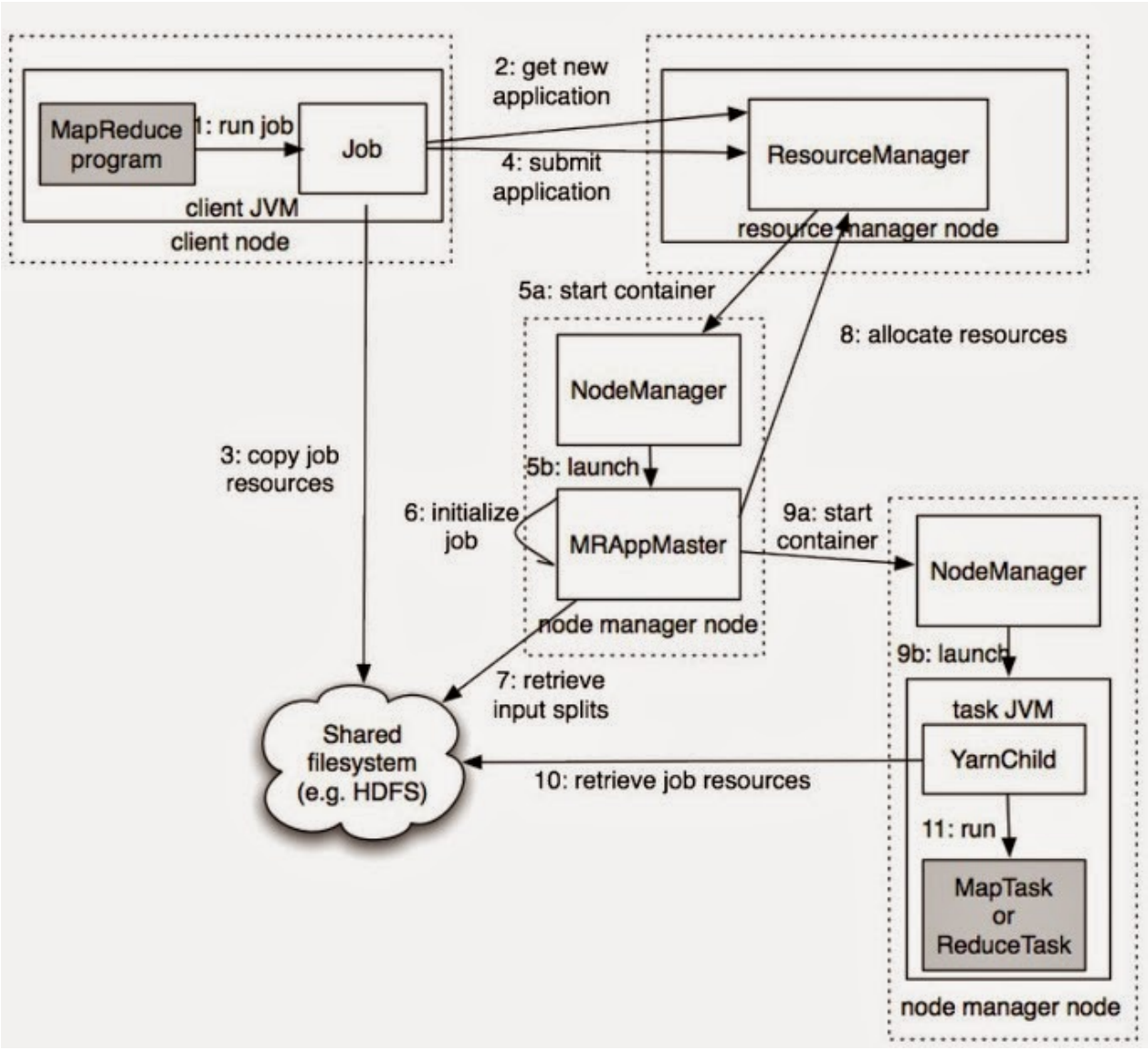


Figure 2: Hadoop Yarn Arch

```
$ vi $HBASE_HOME/conf/hbase-env.sh
export HBASE_HEAPSIZE=16384
```

Step 2. Enable GC logging by using the following command:

```
$ vi $HBASE_HOME/conf/hbase-env.sh
  export HBASE_OPTS="$HBASE_OPTS -verbose:gc -XX:+PrintGCDetails
  -XX:+PrintGCTimeStamps -Xloggc:/usr/local/hbase/logs/gc-hbase.log"
```

Step 3. Add the following code to start the Concurrent-Mark-Sweep GC(CMS) earlier than the default:

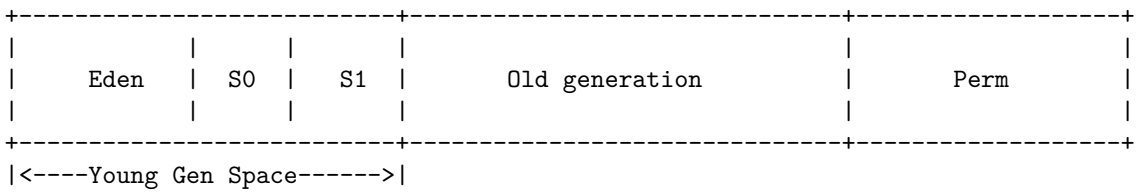
```
$ vi $HBASE_HOME/conf/hbase-env.sh
export HBASE_OPTS="$HBASE_OPTS -XX:CMSInitiatingOccupancyFraction=60"
```

Sync the changes across the cluster and restart HBase.

How it works.

In step 1, we configure the HBase heap memory size. By default, HBase uses a heap size of 1GB, which is too low for modern machines. A heap size of more than 4GB is good for HBase, while our recommendation is 8GB or larger, but under 16GB. Here we set it to 16GB.

In step 2, we enable the JVM logging. With that setting, you will get a region server's JVM logs. Basic knowledge about JVM memory allocation and garbage collection is required to understand the log output.



There are three heap generations: the `Perm(or Permanent) generation` , the `Old Generation` (or Tenured) generation , and the `Young Generation`. The `young generation` section consists of three separate spaces: the `Eden space` and two survivor spaces, `S0` and `S1`.

1. Usually, objects are allocated in the `Eden space` of the `young generation`.
2. If an allocation fails (the Eden is full), all Java threads are halted and a `young generation` GC (Minor GC) is invoked.
3. All surviving objects in the `young generation` (Eden and S0 space) are copied to the `S1` space.
4. If the `S1` space is full, objects are copied (promoted) to the `old generation`.
5. The `old generation` is collected (Major/Full GC) when a promotion fails.
6. The `permanent` and `old generations` are usually collected together.
7. The `permanent generation` is used to hold class and method definitions for objects.

Here is a `GC` Log output.

Though here it does not have any issues and looks to be that our configuration is working. Lets see a scenario where we have some issues, here is one which has some issues.

Before we go into the issue, lets get some information about the thinks which are displayed in the image above.

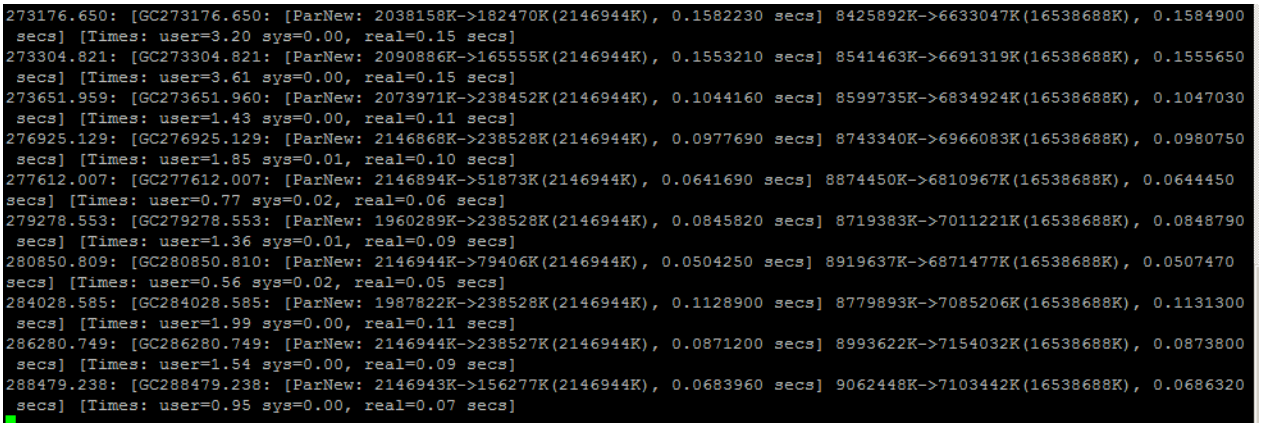


Figure 3: alt text

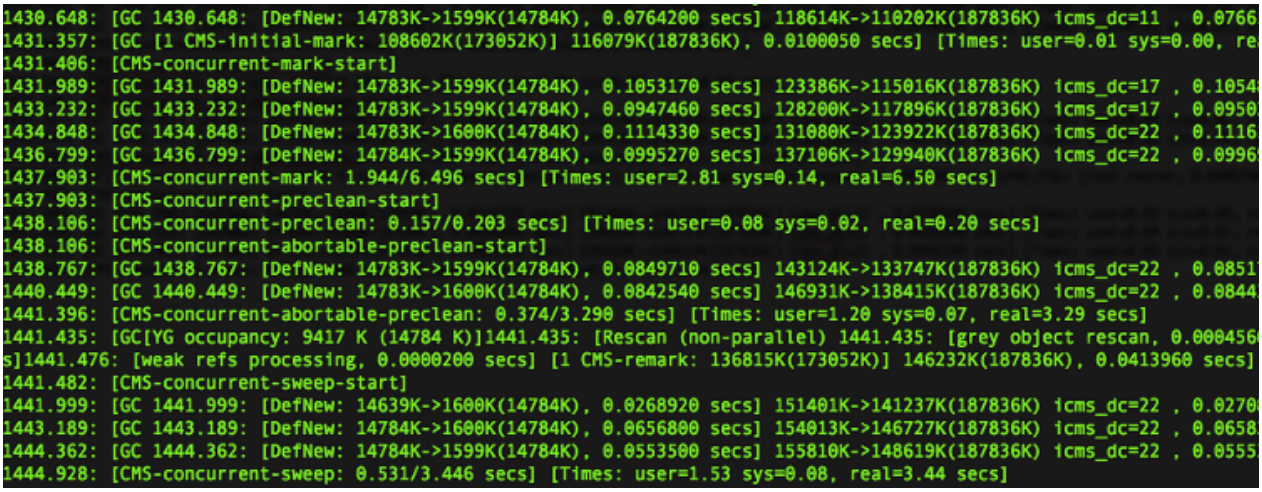


Figure 4: alt text

```
<timestamp>: [GC [<collector>: <starting occupancy1> -> <ending occupancy1>, <pause time1> secs]
<starting occupancy3> -> <ending occupancy3>, <pause time3> secs]
[Times: <user time> <system time>, <real time>]
```

In this output:

1. `<timestamp>` is the times at which the GCs happen, relative to the start of the application.
2. `<collector>` is an internal name for the collector used in the minor collection.
3. `<starting occupancy1>` is the occupancy of the young generation before the collection.
4. `<ending occupancy1>` is the occupancy of the young generation after the collection.
5. `<pause time1>` is the pause time in seconds for the minor collection.
6. `<starting occupancy3>` is the occupancy of the entire heap before the collection.
7. `<ending occupancy3>` is the occupancy of the entire heap after the collection.
8. `<pause time3>` is the pause time for the entire garbage collection. This would include the time for a major collection.
9. `[Time:]` explains the time spend in GC collection, user time, system time, and real time.

Now lets get to the problem

The first line of our output in `second_image` indicates a minor GC, which pauses the JVM for 0.0764200 seconds. It has reduced the young generation space from about 14.8MB to 1.6MB.

Following that, we see the CMS GC logs. HBase uses CMS GC as its default garbage collector for the old generation.

CMS GC performs the following steps :

1. Initial mark
2. Concurrent marking
3. Remark
4. Concurrent sweeping

CMS halts the application's threads only during the initial mark and remark phases. During the concurrent marking and sweeping phases, the CMS thread runs along with the application's threads.

The second line in the example indicates that the CMS initial mark took `0.0100050` seconds and the concurrent marking has taken `6.496` seconds. Note that it is a concurrent marking; Java was not paused.

There is a pause at the line that starts with `1441.435: [GC[YG occupancy:...]` in the earlier screenshot of the GC log. The pause here is `0.0413960` seconds to remark the heap. After that, you can see the sweep starts. The CMS sweep took `3.446` seconds, but the heap size didn't change that much (it kept on occupying about 150MB) here.

The tuning point here is to keep all these pauses low. To keep the pauses low, you may need to adjust the young generation space's size via the `-XX:NewSize` and `-XX:MaxNewSize` JVM flags, to set them to relative small values (for example, up to several hundred MB). If the server has more CPU power, we recommend using the Parallel New Collector by setting the `-XX:+UseParNewGC` option. You may also want to tune the parallel GC thread number for the young generation, via the `-XX:ParallelGCThreads` JVM flag.

We recommend adding the aforementioned settings to the `HBASE_REGIONSERVER_OPTS` variable , instead of the `HBASE_OPTS` variable in the hbase-env.sh file. The `HBASE_REGIONSERVER_OPTS` variable only affects the region server processes, which is good, because the HBase master neither handles heavy tasks nor participates in the data process.

For the old generation, the concurrent collection (CMS) generally cannot be sped up, but it can be started earlier. CMS starts running when the percentage of allocated space in the old generation crosses a threshold. This threshold is automatically calculated by the collector. For some situations, especially during loading, if the CMS starts too late, HBase may run into full garbage collection. To avoid this, we recommend setting the `-XX:CMSInitiatingOccupancyFraction` JVM flag to explicitly specify at what percentage the CMS should be started, as what we did in `step 3`. Starting at 60 or 70 percent is a good practice. When using CMS for an old generation, the default young generation GC will be set to the Parallel New Collector.

Complete Configuration below

```
# The maximum amount of heap to use, in MB. Default is 1000.
export HBASE_HEAPSIZE=16384

# Extra Java runtime options.
# Below are what we set by default.  May only work with SUN JVM.
# For more on why as well as other possible settings,
# see http://wiki.apache.org/hadoop/PerformanceTuning
export HBASE_OPTS="-XX:+UseConcMarkSweepGC"
export HBASE_OPTS="$HBASE_OPTS -verbose:gc -XX:+PrintGCDetails
                            -XX:+PrintGCTimeStamps -Xloggc:/opt/hbase/logs/gc-hbase.log"
export HBASE_OPTS="$HBASE_OPTS -XX:CMSInitiatingOccupancyFraction=60"


# Uncomment and adjust to enable JMX exporting
# See jmxremote.password and jmxremote.access in $JRE_HOME/lib/management to configure remote password a
# More details at: http://java.sun.com/javase/6/docs/technotes/guides/management/agent.html
#

export HBASE_JMX_BASE="-Dcom.sun.management.jmxremote.ssl=false
                                        -Dcom.sun.management.jmxremote.authenticate=false"
export HBASE_MASTER_OPTS="$HBASE_MASTER_OPTS $HBASE_JMX_BASE
                                        -Dcom.sun.management.jmxremote.port=10101"
export HBASE_REGIONSERVER_OPTS="-Xms16G -Xmx16g -XX:+ParallelRefProcEnabled
                                        -XX:-ResizePLAB -XX:ParallelGCThreads=28
                                        $HBASE_REGIONSERVER_OPTS $HBASE_JMX_BASE
                                        -Dcom.sun.management.jmxremote.port=10102"
export HBASE_THRIFT_OPTS="$HBASE_THRIFT_OPTS $HBASE_JMX_BASE
                                        -Dcom.sun.management.jmxremote.port=10103"
export HBASE_ZOOKEEPER_OPTS="$HBASE_ZOOKEEPER_OPTS $HBASE_JMX_BASE
                                        -Dcom.sun.management.jmxremote.port=10104"
export HBASE_REST_OPTS="$HBASE_REST_OPTS $HBASE_JMX_BASE
                                        -Dcom.sun.management.jmxremote.port=10105"
```

Setting `yarn-site.xml` parameters.

Each machine in our cluster has 192 GB of RAM. Some of this RAM should be reserved for Operating System and other services usage. On each node, we'll assign 40 GB RAM for YARN to use and keep rest for the Operating System and other processes. The following property sets the maximum memory YARN can utilize on the node:

In `yarn-site.xml`

```
<name>yarn.nodemanager.resource.memory-mb</name>
<value>40960</value>
```

The next step is to provide YARN guidance on how to break up the total resources available into Containers. You do this by specifying the minimum unit of RAM to allocate for a Container. We want to allow for a maximum of 20 Containers, and thus need (40 GB total RAM) / (20 # of Containers) = 2 GB minimum per container:

In `yarn-site.xml`

```
<name>yarn.scheduler.minimum-allocation-mb</name>
<value>2048</value>
```

YARN will allocate Containers with RAM amounts greater than the `yarn.scheduler.minimum-allocation-mb`.

Hadoop `JAVA_OPTS` Configuration.

Setting `namenode` to 16GB. Since `NN` needs RAM just like `Hbase` dont starve it.

```
# Command specific options appended to HADOOP_OPTS when specified
export HADOOP_NAMENODE_OPTS="-Xms16G -Xmx16g
                                    -Dhadoop.security.logger=${HADOOP_SECURITY_LOGGER:-INFO,RFAS}
                                    -Dhdfs.audit.logger=${HDFS_AUDIT_LOGGER:-INFO,NullAppender}
                                    $HADOOP_NAMENODE_OPTS"
```

Setting `datanode` to 16GB. `datanode` Heap depends on the Server configuration. 4-8GB of Heap space should be fine for `datanode`.

```
export HADOOP_DATANODE_OPTS="-Xms16G -Xmx16g -Dhadoop.security.logger=ERROR,RFAS
                                    $HADOOP_DATANODE_OPTS"
```

Setting `secondary-namenode` same as `namenode`.

```
export HADOOP_SECONDARYNAMENODE_OPTS="-Xms16G -Xmx16g
                                    -Dhadoop.security.logger=${HADOOP_SECURITY_LOGGER:-INFO,RFAS}
                                    -Dhdfs.audit.logger=${HDFS_AUDIT_LOGGER:-INFO,NullAppender}
                                    $HADOOP_SECONDARYNAMENODE_OPTS"
```

More information on changing these Parameter are here How to Plan and Configure YARN

Hadoop/Hbase `sysctl.conf` parameters.

File System > 1. `fs.file-max` Increase size of file handles and inode cache

```
[ahmed@server ~]# echo 'fs.file-max = 943718' >> /etc/sysctl.conf
```

Swappiness : Do less swapping

1. `vm.dirty_ratio` setting virtual memory ratio.
2. `vm.swappiness` How often swap should be used. 0 is least, 60 default.

```
[ahmed@server ~]# echo 'vm.dirty_ratio=10' >> /etc/sysctl.conf
[ahmed@server ~]# echo 'vm.swappiness=0' >> /etc/sysctl.conf
```

Connection Settings

1. `net.core.netdev_max_backlog` Increase number of incoming connections backlog.
2. `net.core.somaxconn` Increase number of incoming connections.

```
[ahmed@server ~]# echo 'net.core.netdev_max_backlog = 4000' >> /etc/sysctl.conf
[ahmed@server ~]# echo 'net.core.somaxconn = 4000' >> /etc/sysctl.conf
```

TCP settings

1. `net.ipv4.tcp_sack` Disable select acknowledgments
2. `net.ipv4.tcp_dsack` Allows TCP to send "duplicate" SACKs.
3. `net.ipv4.tcp_keepalive_time` How often TCP sends out keepalive messages when keepalive is enabled. Default: 2hours.
4. `net.ipv4.tcp_keepalive_probes` How many keepalive probes TCP sends out, until it decides that the connection is broken. Default value: 9.
5. `net.ipv4.tcp_keepalive_intvl` How frequently the probes are send out. Multiplied by `tcp_keepalive_probes` it is time to kill not responding connection, after probes started. Default value: 75sec i.e. connection will be aborted after ~11 minutes of retries.
6. `net.ipv4.tcp_fin_timeout` Time to hold socket in state FIN-WAIT-2, if it was closed by our side. Peer can be broken and never close its side, or even died unexpectedly. Default value is 60sec. Usual value used in 2.2 was 180 seconds, you may restore it, but remember that if your machine is even underloaded WEB server, you risk to overflow memory with kilotons of dead sockets, FIN-WAIT-2 sockets are less dangerous than FIN-WAIT-1, because they eat maximum 1.5K of memory, but they tend to live longer. `Cf.tcp_max_orphans`.
7. `net.ipv4.tcp_rmem` The three values setting the minimum, initial, and maximum size of the Memory Receive Buffer per connection. They define the actual memory usage, not just TCP window size.
8. `net.ipv4.tcp_wmem` The same as tcp_rmem, but just for Memory Send Buffer per connection.
9. `net.ipv4.tcp_retries2` This value influences the timeout of an alive TCP connection, when RTO retransmissions remain unacknowledged. Given a value of N, a hypothetical TCP connection following exponential backoff with an initial RTO of `TCP_RTO_MIN` would retransmit N times before killing the connection at the (N+1)th RTO. The default value of 15 yields a hypothetical timeout of 924.6 seconds and is a lower bound for the effective timeout. TCP will effectively time out at the first RTO which exceeds the hypothetical timeout. RFC 1122 recommends at least 100 seconds for the timeout, which corresponds to a value of at least 8.
10. `net.ipv4.tcp_synack_retries` Number of times SYNACKs for a passive TCP connection attempt will be retransmitted. Should not be higher than 255. Default value is 5, which corresponds to ~180seconds.

```
[ahmed@server ~]# echo 'net.ipv4.tcp_sack = 0' >> /etc/sysctl.conf
[ahmed@server ~]# echo 'net.ipv4.tcp_dsack = 0' >> /etc/sysctl.conf
[ahmed@server ~]# echo 'net.ipv4.tcp_keepalive_time = 600' >> /etc/sysctl.conf
[ahmed@server ~]# echo 'net.ipv4.tcp_keepalive_probes = 5' >> /etc/sysctl.conf
[ahmed@server ~]# echo 'net.ipv4.tcp_keepalive_intvl = 15' >> /etc/sysctl.conf
[ahmed@server ~]# echo 'net.ipv4.tcp_fin_timeout = 30' >> /etc/sysctl.conf
[ahmed@server ~]# echo 'net.ipv4.tcp_rmem = 32768 436600 4194304' >> /etc/sysctl.conf
[ahmed@server ~]# echo 'net.ipv4.tcp_wmem = 32768 436600 4194304' >> /etc/sysctl.conf
[ahmed@server ~]# echo 'net.ipv4.tcp_retries2 = 10' >> /etc/sysctl.conf
[ahmed@server ~]# echo 'net.ipv4.tcp_synack_retries = 3' >> /etc/sysctl.conf
```

Disable IPv6 Defaults. We dont use these anyway.

```
[ahmed@server ~]# echo 'net.ipv6.conf.all.disable_ipv6 = 1' >> /etc/sysctl.conf
[ahmed@server ~]# echo 'net.ipv6.conf.default.disable_ipv6 = 1' >> /etc/sysctl.conf
[ahmed@server ~]# echo 'net.ipv6.conf.lo.disable_ipv6 = 1' >> /etc/sysctl.conf
```

Execute below command to make it permanent.

```
[ahmed@server ~]# sysctl -p
```

Next update limits.

```
[ahmed@server ~]# echo '* - nofile 100000' >> /etc/security/limits.conf
[ahmed@server ~]# echo '* - nproc 100000' >> /etc/security/limits.conf
```

Completed Script below. Run it as `root`.

```
echo "Taking sysctl.conf backup..."
cp /etc/sysctl.conf /etc/sysctl.conf.bkpz

echo "#########################################" >> /etc/sysctl.conf
echo "#########-------------------------#######" >> /etc/sysctl.conf
echo "|         CUSTOM SYSCTL.CONF PARAMS       |" >> /etc/sysctl.conf
echo "#########-------------------------#######" >> /etc/sysctl.conf
echo "#########################################" >> /etc/sysctl.conf

echo 'net.ipv4.tcp_sack = 0' >> /etc/sysctl.conf
echo 'net.ipv4.tcp_dsack = 0' >> /etc/sysctl.conf
echo 'net.ipv4.tcp_keepalive_time = 600' >> /etc/sysctl.conf
echo 'net.ipv4.tcp_keepalive_probes = 5' >> /etc/sysctl.conf
echo 'net.ipv4.tcp_keepalive_intvl = 15' >> /etc/sysctl.conf
echo 'net.ipv4.tcp_fin_timeout = 30' >> /etc/sysctl.conf
echo 'net.ipv4.tcp_rmem = 32768 436600 4194304' >> /etc/sysctl.conf
echo 'net.ipv4.tcp_wmem = 32768 436600 4194304' >> /etc/sysctl.conf
echo 'net.ipv4.tcp_retries2 = 10' >> /etc/sysctl.conf
echo 'net.ipv4.tcp_synack_retries = 3' >> /etc/sysctl.conf
echo 'net.ipv6.conf.all.disable_ipv6 = 1' >> /etc/sysctl.conf
echo 'net.ipv6.conf.default.disable_ipv6 = 1' >> /etc/sysctl.conf
echo 'net.ipv6.conf.lo.disable_ipv6 = 1' >> /etc/sysctl.conf
echo 'vm.dirty_ratio = 10' >> /etc/sysctl.conf
echo 'vm.swappiness = 0' >> /etc/sysctl.conf
echo 'fs.file-max = 943718' >> /etc/sysctl.conf

echo "Making changes Permanenet - Executing `sysctl -p` command..."
sysctl -p

echo "################################################################"
echo "UPDATE COMPLETE"
echo "################################################################"
```

Extras - Monitoring JVM

For setting up JVM monitoring on Zabbix More Details can be found here in zabbix_java_gateway

This below option enables JVM for monitoring.

```
-Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.authenticate=false
```

Below we are setting the jmxport which a remote monitoring script/server can pick up monitoring information about the JVM.

```
-Dcom.sun.management.jmxremote.port=10101
```

More Details on IPv4

```
http://www.cyberciti.biz/files/linux-kernel/Documentation/networking/ip-sysctl.txt
http://www.cyberciti.biz/files/linux-kernel/Documentation/sysctl/
https://www.packtpub.com/books/content/hbase-administration-performance-tuning
https://software.intel.com/en-us/blogs/2014/06/18/part-1-tuning-java-garbage-collection-for-hbase
```

http://stackoverflow.com/questions/22455562/what-set-the-value-of-jvm-parameter-maxnewsize-ergonomics
http://docs.oracle.com/cd/E22289_01/html/821-1274/configuring-the-default-jvm-and-java-arguments.html#so
http://archive.cloudera.com/cdh5/cdh/5/hbase/book/performance.html
http://www-01.ibm.com/support/knowledgecenter/SSPT3X_2.1.2/com.ibm.swg.im.infosphere.biginsights.analyze
http://wiki.apache.org/hadoop/PerformanceTuning
http://hbase.apache.org/book.html#perf.writing