Exercise 04.03: Monitor, Measure, and Diagnose Cluster Performance

In this exercise, you will accomplish the following:

- Use Linux tools to assess cluster performance
- Use nodetool to assess cluster performance
- Use logging to understand performance constraints

Background

There are four (4) major resources that may constrain any node:

- CPU
- Memory
- Disk IO
- Network IO

While running `cassandra-stress` to simulate the projected workload for the cluster, you observed that the stress test eventually saturates the single-node cluster. How can you determine which resource is the constraint?

Step 1: Determine Resource Constraints

1. Confirm the two (2) terminal windows from the previous exercise are still active. One is connected to the instance running the Cassandra node (DSE-node1), and the other is connected to the instance running the test (DSE-node2).
2. In the DSE-node1 window, verify the cluster is up and running.

```
nodetool status
```

3. In the DSE-node2 window, start the cassandra-stress command shown below. Keep cassandra-stress running while performing the remainder of the steps in this exercise.
   a. If cassandra-stress completes before finishing the remaining steps, restart cassandra-stress.

```
cassandra-stress user profile=/home/ubuntu/labwork/TestProfile.yaml
ops\(insert=1,user_by_email=10\) -node DSE-node1
```

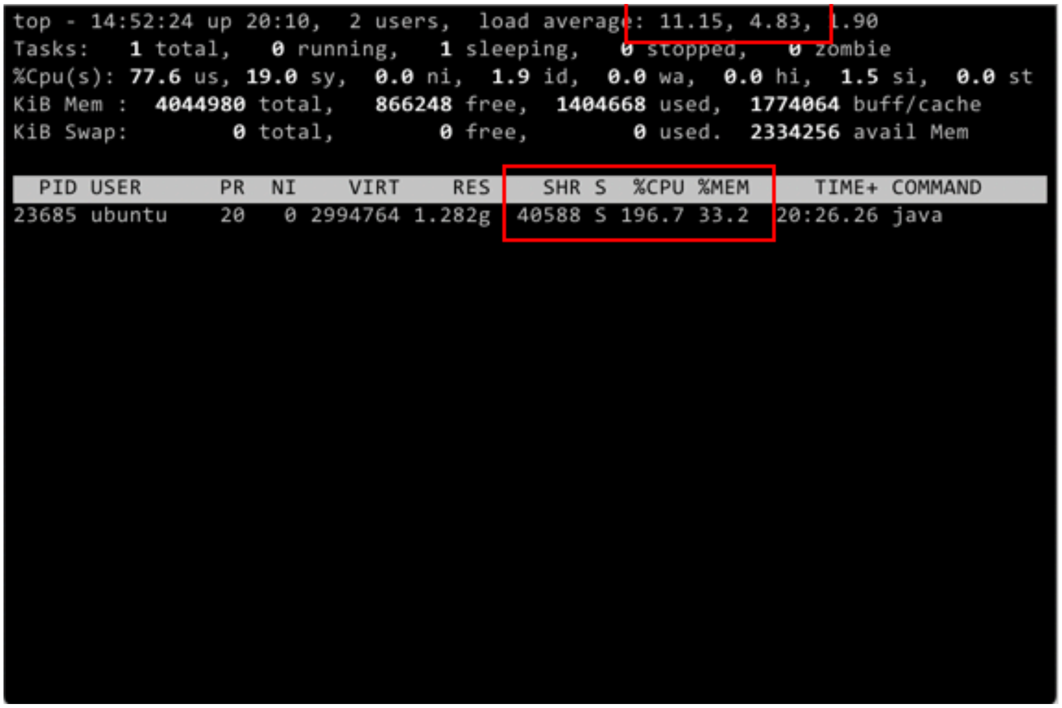
4. First, understand what is happening with the Cassandra node, i.e. node1. Use the `top` command to get a general understanding of node1's behavior. In the DSE-node1 window, find the process ID for the cassandra process using the following command:

```
ps -ef | grep cassandra
```

5. In the DSE-node1 window, run the top command with the `-p` option.

```
ps -ef | grep cassandra
top -p <NODES_CASSANDRA_PROCESS_ID_GOES_HERE>
```

Note how the system resources change as the stress load increases. What do you consider to be the constraining resource and why?



6. While running top, press 1 to see output on a per-CPU basis.
7. Notice that `top` only displays the CPU and memory utilization. Also, note that as the stress load increases due to an increase in the number of test threads, the node's CPU utilization may also increase. If you continue to observe `top` as the test workload saturates the node, you may notice that the CPU exceeds 100% utilization (with two (2) cores you could reach as much as 200%).
    a. This is evidence that the CPU is the constraining resource. 100% is the limit because the machine is a 2-core machine, and one of the cores is reserved for other work such as garbage collection or compaction.
8. Examine the disk and network utilization. How is that done? In the DSE-node1 window exit out of `top` by typing `control-C` and use the `dstat` command to see what else is happening with other resources.

```
dstat -am
```

9. Watch the `dstat` output; as the load increases you will note that the CPU is saturated (slightly over 50%). At the same time, notice that memory is not fully utilized. You can also observe that the disk read and write activity is small relative to disk capabilities. You also see that the network traffic is negligible. Notice also there is no paging – paging would be a sign that something was seriously askew. This should strengthen the supposition that the CPU is the constraining resource.
   a. When done studying the `dstat` output, quit `dstat` by typing `control-C`.

```
  0   0 100   0   0   0|   0     0 |  52B  416B|   0     0 | 214   389 >
  0   0 100   0   0   0|   0     0 |  52B  358B|   0     0 | 206   375 >
  0   0 100   0   0   0|   0     0 |  52B  358B|   0     0 | 197   365 >
  0   0 100   0   0   0|   0     0 |  52B  358B|   0     0 | 203   374 >
  0   0 100   0   0   0|   0     0 |  52B  358B|   0     0 | 195   359 >
 16   0  85   0   0   0|   0     0 | 165B  606B|   0     0 | 435   706 >
 22   5  72   0   0   1|   0     0 | 573k  547k|   0     0 |2823  6053 >
 81  19   1   0   0   0|   0     0 |1842k 1962k|   0     0 |3329  8010 >
 80  17   4   0   0   0|   0   976k|1863k 1946k|   0     0 |4922   12k>
 78  21   1   0   0   1|   0     0 |1938k 2044k|   0     0 |3677  8579 >
 78  19   2   0   0   1|   0     0 |1908k 1972k|   0     0 |4116   10k>
 80  18   2   0   0   1|   0     0 |1908k 2008k|   0     0 |3887   11k>
 70  17  13   0   0   0|   0     0 |1684k 1810k|   0     0 |4221   11k>
 81  17   2   0   0   1|   0     0 |1812k 1848k|   0     0 |3960   13k>
 81  18   1   0   0   1|   0     0 |1925k 1978k|   0     0 |4365   10k>
 76  22   1   0   0   1|   0     0 |1988k 2018k|   0     0 |4610  8894 >
 80  19   1   0   0   1|   0     0 |1929k 2036k|   0     0 |3724   11k>
 80  18   2   0   0   1|   0     0 |1928k 1961k|   0     0 |5091   11k>
 82  17   1   0   0   1|   0  5096k|1895k 2000k|   0     0 |5225  8272 >
----total-cpu-usage---- -dsk/total- -net/total- ---paging-- ---system-->
usr sys idl wai hiq siq| read  writ| recv  send|  in   out | int   csw >
 80  18   1   0   0   1|   0     0 |1910k 2011k|   0     0 |4128  9142 >
 82  18   1   0   0   0|   0     0 |1925k 2041k|   0     0 |3459  8600 >
 81  18   1   0   0   1|   0     0 |1932k 2041k|   0     0 |3484  8070 >
 81  18   2   0   0   1|   0     0 |1943k 1976k|   0     0 |4563  9786 >
```

10. Take time to look at some other characteristics of your node. In the DSE-node1 window, run the `nodetool info` command.

```
nodetool info
```

This won't necessarily help with performance tuning, but it provides a chance to become familiar with the info output. Look at each row of the output. Do you understand what each label means? Do the values associated with the labels make sense?

11. In the DSE-node1 window, run the command `nodetool compactionhistory` and examine the compaction activity.

```
nodetool compactionhistory
```

12. Again, confirm the output makes sense by considering what each column means and what the columns' values are. Does the compaction history seem reasonable?

Step 2: Examine Statistics

1. In the DSE-node1 window, run the command `nodetool gcstats`. Examine each of the columns to make certain you understand what each means. Since `gcstats` only reports measurements since the previous invocation of `gcstats`, run the command several times.

```
nodetool gcstats
```

2. In the DSE-node1 terminal window, try running the command `nodetool gossipinfo`. For a normally functioning node, *gossipinfo* probably won't reveal much about the node's performance. Also, for a one-node cluster, this info is not very interesting. But for a malfunctioning multi-node cluster, it may be helpful to know how each node sees the cluster. Look at each of the rows of the output to make sure you understand them.

```
nodetool gossipinfo
```

3. In the DSE-node1 window, run the command `nodetool ring` for the 'killr_video' keyspace. Once again, this is not very interesting for a single-node cluster, but it provides a chance to familiarize yourself with the output format. Notice that the load column values match the load value `nodetool info` reports.

```
nodetool ring -- killr_video
```

4. In the DSE-node1 window, run the command `nodetool tablestats`. If you do not specify a keyspace and table, the command provides stats for all tables including the system tables. This is a lot of output, so specify the 'killr_video.user_by_email' table. Also, use the `-H` option to make the output a little friendlier.

```
nodetool tablestats -H -- killr_video.user_by_email
```

5. This command has lots of useful information for performance tuning. See https://docs.datastax.com/en/dse/6.7/dse-admin/datastax_enterprise/tools/nodetool/toolsTablestats.html for an explanation of each row of output.
6. In the DSE-node1 window, run `nodetool tablehistograms`. Specify the 'killr_video' keyspace and the 'user_by_email' table. This command displays the disk I/O latencies for a specific table.

```
nodetool tablehistograms killr_video user_by_email
```

7. In the DSE-node1 window, run `nodetool tpstats`. This command shows the thread-pool activity. If you run this command several times, you may see the active column change as thread actively services requests. Pending indicates that requests are queuing up for the thread pool.

```
nodetool tpstats
```

Step 3: Review Logs

1. In the DSE-node1 window, use tail to examine Cassandra's log file, i.e., the debug.log file. Use the `-f` option to see the end of the file as Cassandra writes to it. Remember, the log files are in */home/ubuntu/dse/logs*. Also, remember that the file *system.log* only records INFO and above. Let's examine the *debug.log* file so we can see lower level messages. When ready, use `control-C` to end `tail -f`, but don't stop tailing the logs just yet.

```
tail -f /home/ubuntu/dse/logs/debug.log
```

2. While continuing to watch the tail of the log file in the DSE-node1 window, use a third terminal window to inspect the current logging levels using `nodetool getlogginglevels`. Connect to the DSE-node1 instance and get the logging levels as follows:

```
nodetool getlogginglevels
```

Notice that the *org.apache.cassandra* is set to `DEBUG`.

3. Change the logging level of *org.apache.cassandra* to TRACE and see what happens to the log file. In the third terminal window, use `nodetool setlogginglevel` to change `org.apache.cassandra`. Make sure you made the change correctly by re-inspecting the logging levels.

```
nodetool setlogginglevel org.apache.cassandra TRACE
nodetool getlogginglevels
```

4. What is do you see happening to the *debug.log* file? How might this information prove useful at some later time?

5. In the third terminal window, set the log level back to DEBUG. After you do, check the logging levels to make sure your change worked as expected. You should also notice the logging to the *debug.log* file calms down as soon as the change takes effect.

```
nodetool setlogginglevel org.apache.cassandra DEBUG
nodetool getlogginglevels
```

6. Delete the third terminal window but leave the other two (2) (i.e., DSE-node1 and DSE-node2) windows open for use in the next exercise.

7. In the DSE-node2 window, allow cassandra-stress to end normally. Make note of the summary statistics for the test. Since we believe the node is CPU constrained, what do you think the effects of additional CPU resources would be on the statistics?

END OF EXERCISE