CPU and Memory Load Testing of Spark Servers

This document describes some basic CPU and memory load testing on the Spark servers used for the CERN Hadoop Service in April 2023. It reports on the tests performed, tools used, findings, and lessons learned.

Author and contact: Luca.Canali@cern.ch

Motivations

CPU usage is important for data processing: We observe that workloads using Spark on the Hadoop services at CERN are often CPU-bound. Notably, we observe that when Apache Spark has to process large Parquet files, the operations require several GBs of RAM often core and are often CPU-bound. It is important for the evolution of the Spark service (Spark on Hadoop and Spark on Kubernetes) to understand the CPU demands from users' jobs and to understand what the available HW is capable of delivering. This exercise provides some test data for two of the recent data platforms used by our services.

Description and limitations of the tests

The tests reported here are quite limited in scope, as they focus only on CPU and performance and with two specific and "narrow" workloads. However, I believe they provide some indications on the behavior of the server CPU performance and the overall CPU capacity of the tested servers. The comparison between two different server models is the original motivator for this work, in particular the interest was to understand those servers can perform and scale when used for running Apache Spark and Hadoop processed. Note, this work is not intended to be a benchmark of the tested systems.

Tools used for load testing

- The first workload generator and testing tool is a simple script burning CPU cycles in a loop and executed using multiple workers running in parallel, two implementations have been used, one in Python and one in Rust, deploying compiled binaries. Both provide similar results.
- The second workload generator is a custom tool that runs on top of Apache Spark. It is meant for stressing specifically CPUs and the CPU-to-memory throughput.

Links to the code, measured data, and data analyses using notebooks:

CPU load testing kit - Python version	Kit for load testing and measuring CPU-intensive workloads, Python version.

CPU load testing kit - Rust version	Kit for Load testing and measuring CPU-intensive workloads, Rust version.
Spark_CPU_memory_load_testkit	Load testing CPU and memory using Apache Spark.

Key findings

The newer server model (RAC55) outperformed the older model (HDP6) in terms of CPU performance for up to 16 concurrent workers. However, due to its higher core count, the HDP6 model demonstrated higher throughput under heavier load.

Test 1 – Concurrent workers burning CPU cycles in a loop: The RAC55 server model demonstrated faster per-thread CPU performance up to 16 concurrent threads. The difference in performance between HDP6 (older) and RAC55 (newer) was roughly a 1.3x increase in per-CPU thread performance in favor of RAC55. However, the HDP6 model, with a higher number of cores, provided higher total throughput at saturation, about 1.7x higher than RAC55.

Test 2 – Parallel workers running CPU- and Memory-intensive load: The RAC55 server model displayed the highest performance (1.3x over HDP6) for low load (number of concurrent workers/tasks <= 20). However, as in Test 1, the HDP6 server model, with more cores, demonstrated higher total throughput when the load was increased.

Both server models showed an increase in memory throughput as the load (number of concurrent tasks) increased. The throughput saturation appeared when the number of workers exceeded the number of cores on the server and was finally reached when the number of parallel workers equaled the number of logical cores on the system. The RAC55 model showed the highest performance for low load (number of concurrent workers/tasks <= 20).

Description of the platforms

CPU load tests have been performed on two dedicated test servers representative of production servers, we will refer to them in this document as RAC55 and HDP6.

The servers were installed with RHEL 7.9 and tests with Apache Spark used Spark version 3.3.2

We omit the configuration of networking and I/O, as not relevant for these tests.

We don't report the exact CPU models in this doc.

RAC55:

- 16 physical cores (2 sockets, 8 physical cores each), 32 logical cores visible on the OS due to hyperthreading
- CPU nominal freq: 3.7 GHz
- CPU from 2019, CPU architecture: Zen 3
- L1 caches: 32K + 32K, L2 cache 512K, L3 cache 32768K
- RAM: DDR4, 1 TB

HDP6:

- 32 physical cores (2 sockets, 16 physical cores each), 64 logical cores visible on the OS due to hyperthreading
- CPU max freq: 3.0 GHz
- CPU from 2017, CPU architecture: Zen 2
- L1 caches: 32K + 32K, L2 cache 512K, L3 cache 16384K
- RAM: DDR4, 512 GB

Test 1 – Concurrent workers burning CPU cycles in a loop

The workload generator and testing tool is a simple Python script (or Rust program) burning CPU cycles in a loop.

The script is executed running on a configurable number of concurrent workers. The script measures the time spent executing a simple CPU-burning loop.

This provides a simple way to generate CPU load on the system.

Example of how the data was collected with the testing tool written in Rust and compiled to binary:

```
./test_cpu_parallel --num_workers 8 --full --output myout.csv
```

See code at https://github.com/LucaCanali/Miscellaneous/tree/master/Performance_Testing

The advantage of this approach is that the testing tool is easy to write and can be easily automated.

The weak point of testing this way is that the test workload is somewhat “artificial” and disconnected with the server actual purpose as a DB server. For example, the CPU-burning loop used for this test is mostly instruction-intensive on the CPU and does not spend much time on memory access.

Measurements and results:

The following figures represent the same data in different ways to highlight different performance and scalability characteristics.

Figure 1 – Raw data

- The figure reports the testing job execution time, measured for varying server load on the three tested servers.
- A common pattern is that at low load (see data with just a few parallel workers) the job run time is almost constant.
- An important difference is that the job run time is different on the different platforms, with RAC55 being faster than HDP6.
- Another pattern is that the job running starts to increase linearly at higher load.
- The job execution time curve starts to bend upwards as the load increases. Typically, we see this happening when the num of parallel workers is greater than the number of physical cores on the server (16 cores on RAC55, 32 cores on HDP6)

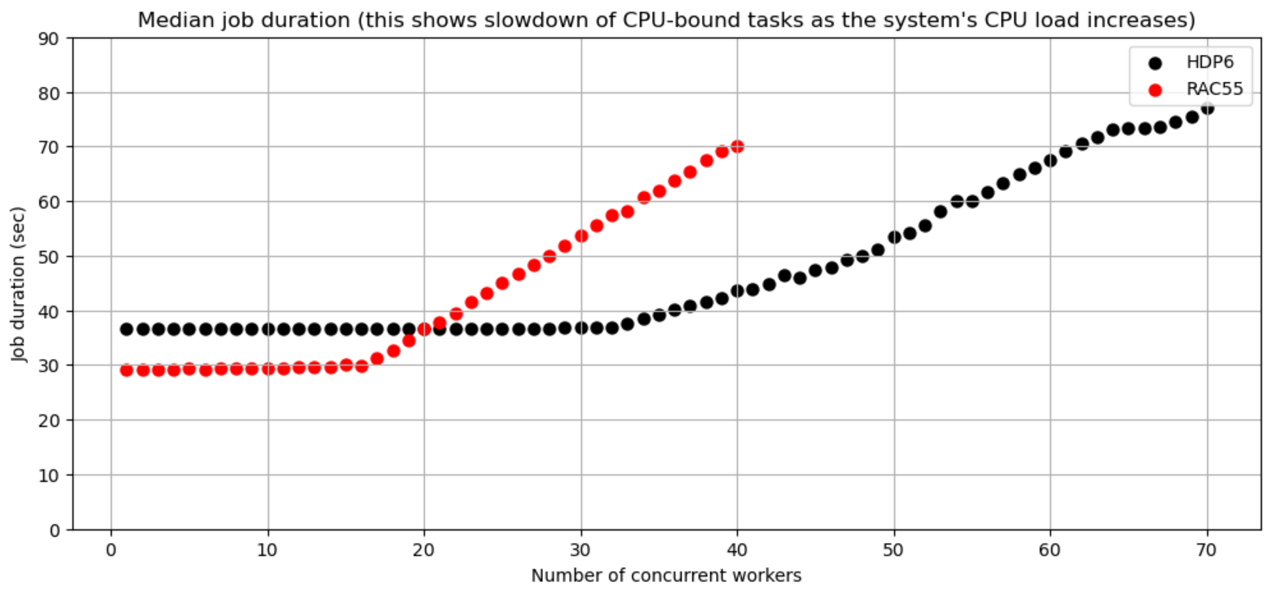


Figure 2 - Speed

- This plot reports the number of jobs per minute per worker
- Data points can be interpreted as a measure of the “speed of the CPU” for a new job coming into the system given a defined system load
- We see that the “effective CPU speed” decreases as the load increases, with sudden changes at the points where the number of parallel workers is equal to the number of physical cores
- The CPU speed per thread is also different depending on the CPU architecture, with RAC55 being faster than HDP6.

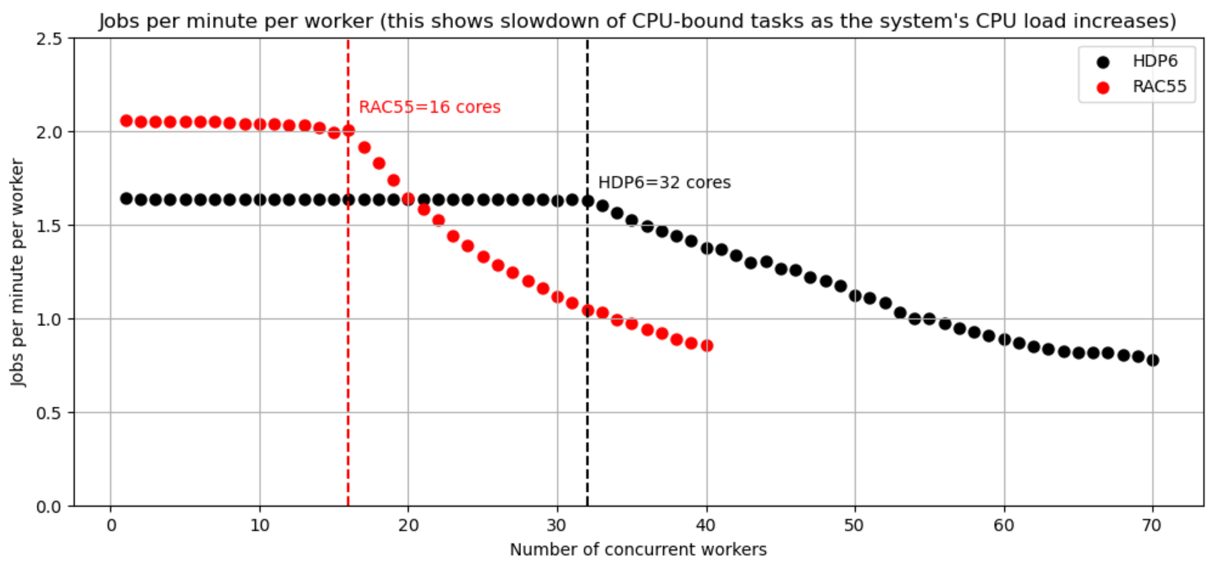


Figure 3 - Capacity

- This plot shows the number of jobs executed per minute summed over all the running worker threads.
- As the load increases the server capacity increases, reaching a maximum value roughly when the number of workers = number of logical cores.
- This allows to compare the “Total CPU capacity” of the two servers. HDP6 has the highest number of cores and the highest CPU throughput at saturation.

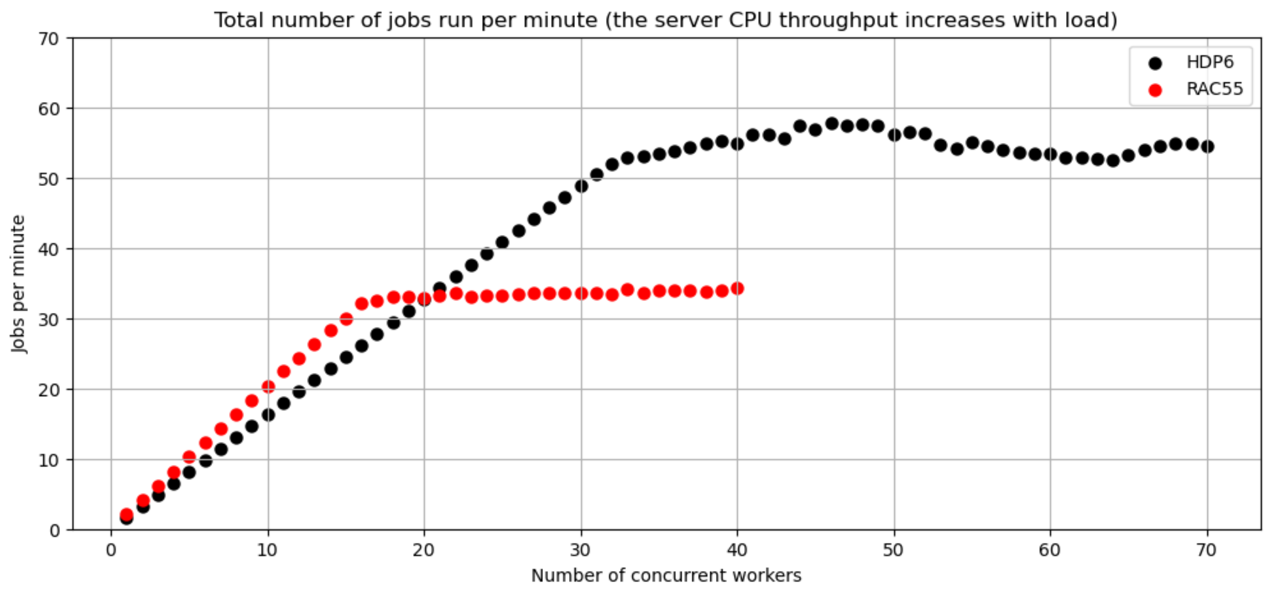
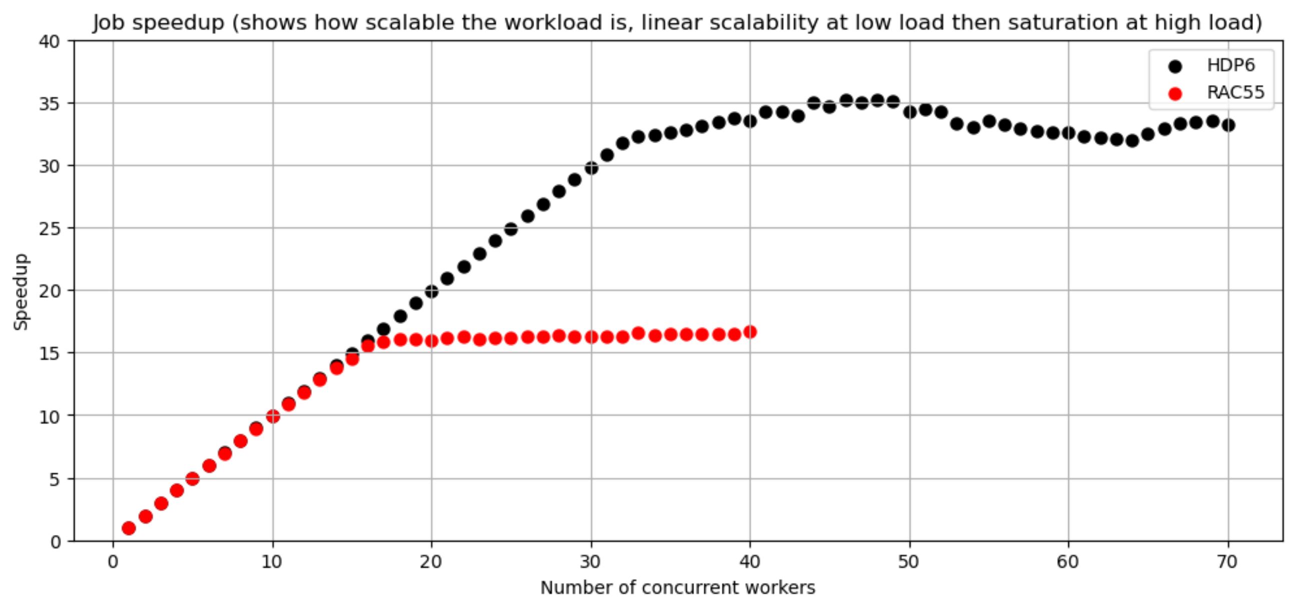


Figure 4 - Scalability

- This shows the speedup, a measure of scalability, for this plot it’s calculated as the ration of the N * (job execution time at load n) / (job execution time at load 1)
- We see almost linear scalability for low loads (up to the number of physical cores), then a slower increase up to the number of logical cores and, finally, saturation
- Both HDP6 and RAC55 appear to scale almost linearly up to the number of physical cores (respectively 32 and 16)



Notes:

- Of the tested servers RAC55 appears the fastest on per-thread CPU performance, up to the 16 concurrent threads (16 is the number of physical cores in RAC55).
    - The difference in performance between HDP6 (oldest) and RAC55 (newest) is roughly x1.3 in per-CPU thread performance in favor of RAC55.
- HDP6 has the highest number of cores which provides for higher total throughput at saturation: about 1.7x higher on HDP6 compared to RAC55.

Test 2 – Parallel workers running CPU- and Memory-intensive load

The second workload generator is a custom tool designed for conducting CPU and CPU-to-memory bandwidth load testing. The workload is implemented in Python using PySpark and is designed to be CPU- and memory-intensive. It involves executing a Spark job that reads a large Parquet table in parallel, utilizing a user-defined number of parallel workers. The primary output of the tool is the measurement of the job execution time, which is recorded as a function of the number of parallel workers employed. Running the program in full mode initiates a range of tests and generates a CSV file that contains the recorded values.

These tests take longer to run the simple "tests with a CPU-burning script" described above, moreover they require some expertise with running Apache Spark to configure and validate the test results.

Measurements and results:

The following figures represent the same data in different ways to highlight different performance and scalability characteristics.

Notebooks with the graphs at
https://github.com/LucaCanali/Miscellaneous/tree/master/Performance_Testing/Spark_CPU_memory_load_testkit

Figure 5 – Raw Data and job curation

- The figure shows the measured time to complete the job (reading a large table) as a function of the number of concurrent workers (Spark tasks)
- The common trend is that the job runtime decreases as there are more concurrent workers.
- Measurements are “noisy” so we should take about 10% as error margin on the collected data points.
- There are differences in performance and total throughput with RAC55 being fastest at low load (load <= 16 concurrent workers, as RAC55 has 16 physical cores)

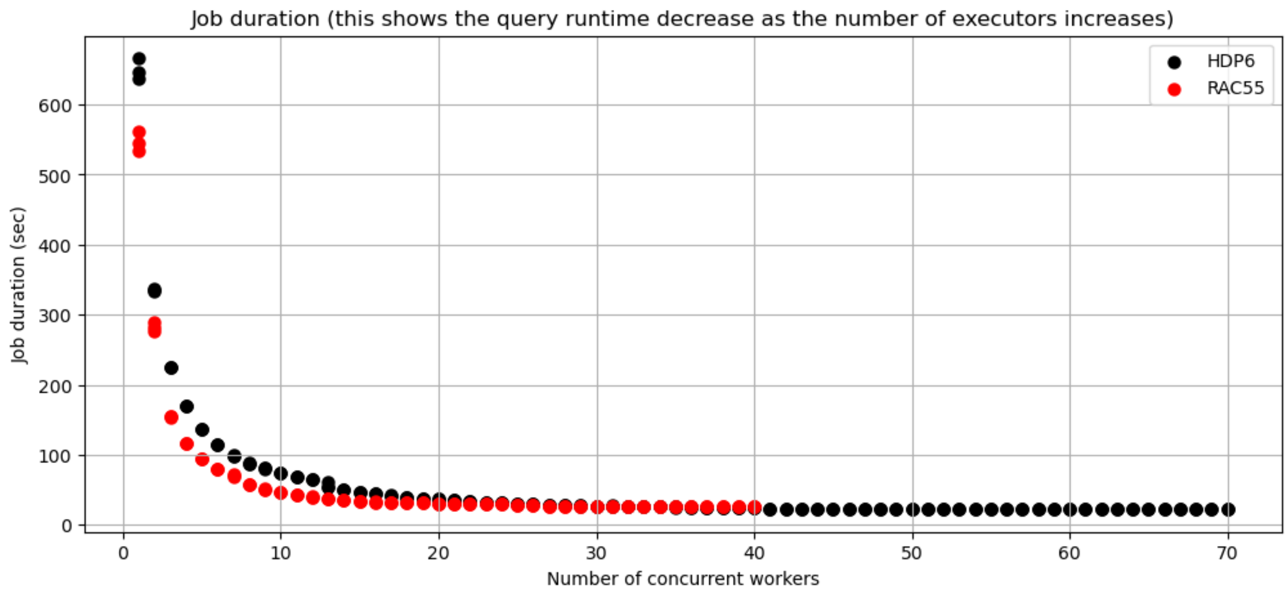
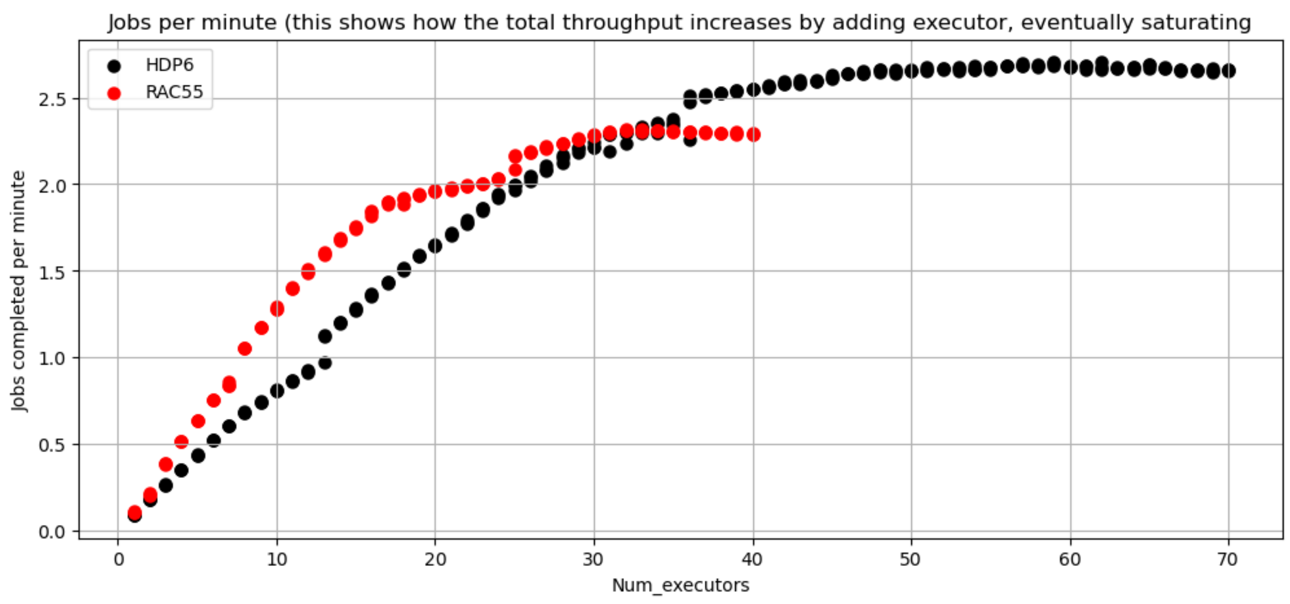


Figure 6 – Throughput, jobs per minute

- The figure shows the number of jobs completed per unit time, a measure of the system’s throughput.
- Measurements are “noisy” so we should take about 10% as error margin on the collected data points.
- The throughput increases as the number of parallel workers increases.
- Saturation in the throughput starts to appear when the number of workers exceeds the number of cores in the server (16 for RAC55 and 32 for HDP6) and is finally reached when the number of parallel workers are equal to the number of logical cores on the system (32 for RAC55 and 64 for HDP6).
- RAC55 shows the highest performance (1.3x over HDP6) for low load (number of concurrent workers/tasks <= 20).



Memory throughput measurements from the OS:

Memory throughput has been monitored and measured while running the Spark-based memory-intensive workload.

In particular, mmeasurements were taken on AMD systems using AMD uProf ("MICRO-prof") tool:
`/opt/AMDuProf_4.0-341/bin/AMDuProfPcm -m memory -a -d 20 -C -A system`

We could find that memory throughput increases as the load (number of concurrent tasks) increases. Memory throughput saturation starts to appear when the number of workers exceeds the number of cores in the server (16 for RAC55 and 32 for HDP6) and is finally reached when the number of parallel workers is equal to the number of logical cores on the system (32 for RAC55 and 64 for HDP6).

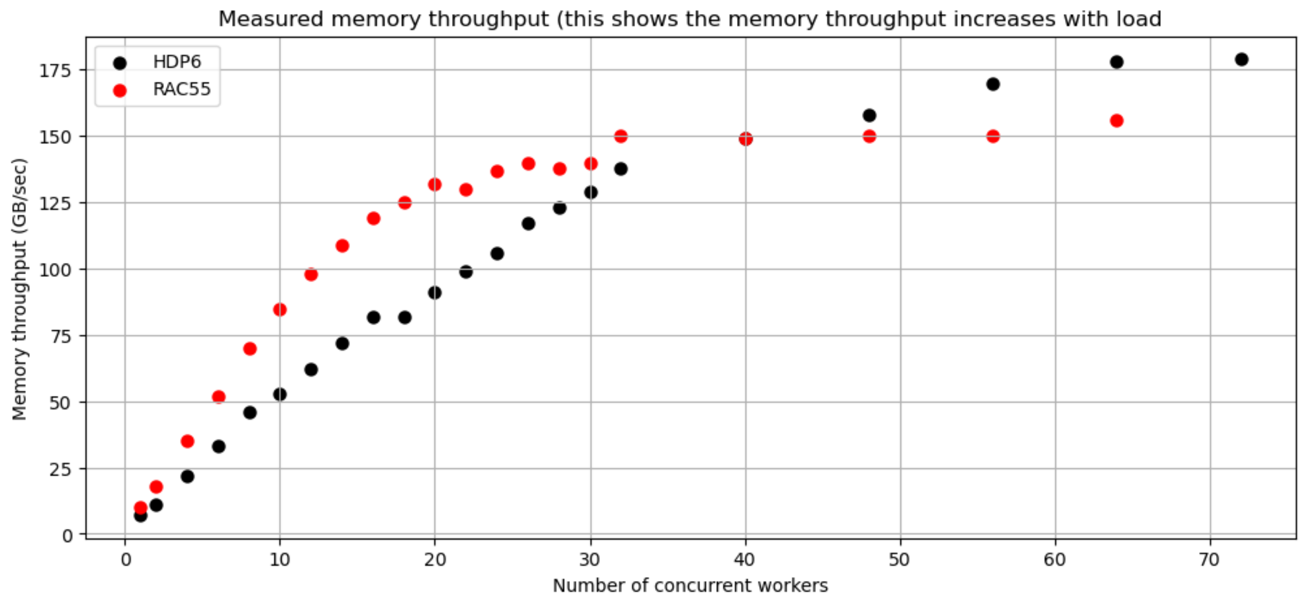
Note that the memory throughput values measured are quite high, however they do not reach HW saturation. With tools described in https://github.com/LucaCanali/Miscellaneous/blob/master/Performance_Testing/Tools_Linux_Memory_Perf_Measure.md we could find that RAC55 was capable of performing high-load tests with an aggregated memory throughput of 260 GB/sec and HDP6 of 220 GB/sec.

Figure 7 – Measured memory throughput

- The figure shows the measured memory throughput while running the Spark-based load testing kit described above.
- Measurements are "noisy" so we should take about 10% as error margin on the collected data points.
- The throughput increases as the number of parallel workers increases.
- Similarly to the case shown in the throughput graph (Figure 6), memory throughput saturation starts to appear when the number of workers exceeds the number of cores in the server (16 for

RAC55 and 32 for HDP6) and is finally reached when the number of parallel workers is equal to the number of logical cores on the system (32 for RAC55 and 64 for HDP6).

- RAC55 shows the highest performance (1.3x over HDP6) for low load (number of concurrent workers/tasks <= 20).



Conclusions

This work collects a few tests and measurement on stress testing and CPU loading two different platforms of interest for the CERN databases, Hadoop, and Spark services.

The tests performed are narrow in scope, just addressing the CPU- and memory-intensive loads.

Two different testing tools have been used for these tests: testing with a simple CPU-burning script loop run in parallel, and testing with a workload generator for CPU and memory intensive work using Apache Spark

In both cases we find that the newest server model (RAC55) has the highest CPU performance up to the load of 16 concurrent workers, while the server HPD6 has the highest throughput due to the higher core count (there are 32 cores in HDP6 and 16 cores in RAC55).