Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich


# Cloud Computing Architecture

Semester project report

**Group XXX**
Author 1 - Legi-NR
Author 2 - Legi-NR
Robin Oester - 19-932-557

Systems Group
Department of Computer Science
ETH Zurich
May 6, 2023

## Instructions

- **Please do not modify the template!** Except for adding your solutions in the labeled places, and imputing your group number, names and legi-NR on the title page.

  **Divergence from the template can lead to subtraction of points.**

- Remember to follow the instructions in the project description regarding which files you have to submit.

- Remove this page before generating the final PDF.

## Part 3 [34 points]

1. **[17 points]** With your scheduling policy, run the entire workflow **3 separate times**. For each run, measure the execution time of each PARSEC job, as well as the latency outputs of memcached running with a steady client load of 30K QPS. For each PARSEC application, compute the mean and standard deviation of the execution time [1] across three runs. Also compute the mean and standard deviation of the total time to complete all jobs. Fill in the table below. Finally, compute the SLO violation ratio for memcached for the three runs; the number of data points with 95th percentile latency $>$ 1ms, as a fraction of the total number of data points while the jobs are running.

   **Answer:**

   Create 3 bar plots (one for each run) of memcached p95 latency (y-axis) over time (x-axis) with annotations showing when each PARSEC job started and ended, also indicating the machine they are running on. Using the augmented version of mcperf, you get two additional columns in the output: `ts_start` and `ts_end`. Use them to determine the width of the bar while the height should represent the p95 latency. Align the $x$ axis so that $x = 0$ coincides with the starting time of the first container. Use the colors proposed in this template (you can find them in `main.tex`). For example, use the vips color to annotate when vips started.
   .

| job name | mean time [s] | std [s] |
|---|---|---|
| blackscholes | | |
| canneal | | |
| dedup | | |
| ferret | | |
| freqmine | | |
| radix | | |
| vips | | |
| total time | | |

   **Plots:**

2. **[17 points]** Describe and justify the "optimal" scheduling policy you have designed.

   - Which node does memcached run on? Why?
     **Answer:**

   - Which node does each of the 7 PARSEC jobs run on? Why?
     **Answer:**
     - blackscholes:
     - canneal:
     - dedup:
     - ferret:

[1] You should only consider the runtime, excluding time spans during which the container is paused.

  - freqmine:
  - radix:
  - vips:

- Which jobs run concurrently / are colocated? Why?
  **Answer:**

- In which order did you run 7 PARSEC jobs? Why?
  **Order (to be sorted)**: blackscholes, canneal, dedup, ferret, freqmine, radix, vips
  **Why**:

- How many threads have you used for each of the 7 PARSEC jobs? Why?
  **Answer:**
  - blackscholes:
  - canneal:
  - dedup:
  - ferret:
  - freqmine:
  - radix:
  - vips:

- Which files did you modify or add and in what way? Which Kubernetes features did you use?
  **Answer:**

- Describe the design choices, ideas and trade-offs you took into account while creating your scheduler (if not already mentioned above):
  **Answer:**

Please attach your modified/added YAML files, run scripts, experiment outputs and report as a zip file.

**Important: The search space of all possible policies is exponential and you do not have enough credits to run all of them. We do not ask you to find the policy that minimizes the total running time, but rather to design a policy that has a reasonable running time, does not violate the SLO, and takes into account the characteristics of the first two parts of the project.**

## Part 4 [76 points]

1. **[20 points]** Use the following `mcperf` command to vary QPS from 5K to 125K in order to answer the following questions:

```
$ ./mcperf -s INTERNAL_MEMCACHED_IP --loadonly
$ ./mcperf -s INTERNAL_MEMCACHED_IP -a INTERNAL_AGENT_IP  \
           --noload -T 16 -C 4 -D 4 -Q 1000 -c 4 -t 5 \
           --scan 5000:125000:5000
```

a) [10 points] How does memcached performance vary with the number of threads ($T$) and number of cores ($C$) allocated to the job? In a single graph, plot the 95th percentile latency (y-axis) vs. QPS (x-axis) of memcached (running alone, with no other jobs collocated on the server) for the following configurations (one line each):

- Memcached with $T$=1 thread, $C$=1 core
- Memcached with $T$=1 thread, $C$=2 cores
- Memcached with $T$=2 threads, $C$=1 core
- Memcached with $T$=2 threads, $C$=2 cores

Label the axes in your plot. State how many runs you averaged across (we recommend three runs) and include error bars. The readability of your plot will be part of your grade.

**Plots:**

What do you conclude from the results in your plot? Summarize in 2-3 brief sentences how memcached performance varies with the number of threads and cores.

**Summary:**

b) [2 points] To support the highest load in the trace (125K QPS) without violating the 1ms latency SLO, how many memcached threads ($T$) and CPU cores ($C$) will you need?

**Answer:**

c) [1 point] Assume you can change the number of cores allocated to memcached dynamically as the QPS varies from 5K to 125K, but the number of threads is fixed when you launch the memcached job. How many memcached threads ($T$) do you propose to use to guarantee the 1ms 95th percentile latency SLO while the load varies between 5K to 125K QPS?

**Answer:**

d) [7 points] Run memcached with the number of threads $T$ that you proposed in (c) and measure performance with $C = 1$ and $C = 2$. Use the aforementioned `mcperf` command to sweep QPS from 5K to 125K.

Measure the CPU utilization on the memcached server at each 5-second load time step.

Plot the performance of memcached using 1-core ($C = 1$) and using 2 cores ($C = 2$) in **two separate graphs**, for $C = 1$ and $C = 2$, respectively. In each graph, plot QPS on the x-axis, ranging from 0 to 130K. In each graph, use two y-axes. Plot the 95th percentile latency on the left y-axis. Draw a dotted horizontal line at the 1ms latency SLO. Plot the CPU utilization

(ranging from 0% to 100% for $C = 1$ or 200% for $C = 2$) on the right y-axis. For simplicity, we do not require error bars for these plots.

**Plots:**

2. **[17 points]** You are now given a dynamic load trace for memcached, which varies QPS randomly between 5K and 100K in 10 second time intervals. Use the following command to run this trace:

```
$ ./mcperf -s INTERNAL_MEMCACHED_IP --loadonly
$ ./mcperf -s INTERNAL_MEMCACHED_IP -a INTERNAL_AGENT_IP \
           --noload -T 16 -C 4 -D 4 -Q 1000 -c 4 -t 1800 \
           --qps_interval 10 --qps_min 5000 --qps_max 100000
```

Note that you can also specify a random seed in this command using the `--qps_seed` flag.

For this and the next questions, feel free to reduce the mcperf measurement duration (`-t` parameter, now fixed to 30 minutes) as long as you have at the end at least 1 minute of memcached running alone.

Design and implement a controller to schedule memcached and the PARSEC benchmarks on the 4-core VM. The goal of your scheduling policy is to successfully complete all PARSEC jobs as soon as possible without violating the 1ms 95th percentile latency for memcached. **Your controller should not assume prior knowledge of the dynamic load trace. You should design your policy to work well regardless of the random seed.** The PARSEC jobs need to use the native dataset, i.e., provide the option `-i native` when running them. Also make sure to check that all the PARSEC jobs complete successfully and do not crash. Note that PARSEC jobs may fail if given insufficient resources.

Describe how you designed and implemented your scheduling policy. Include the source code of your controller in the zip file you submit.

- Brief overview of the scheduling policy (max 10 lines):
  **Answer:**

- How do you decide how many cores to dynamically assign to memcached? Why?
  **Answer:**

- How do you decide how many cores to assign each PARSEC job? Why?
  **Answer:**
  - blackscholes:
  - canneal:
  - dedup:
  - ferret:
  - freqmine:
  - radix:

  - vips:

- How many threads do you use for each of the PARSEC job? Why?
  **Answer:**
  - blackscholes:
  - canneal:
  - dedup:
  - ferret:
  - freqmine:
  - radix:
  - vips:

- Which jobs run concurrently / are collocated and on which cores? Why?
  **Answer:**

- In which order did you run the PARSEC jobs? Why?
  **Order** (to be sorted): blackscholes, canneal, dedup, ferret, freqmine, radix, vips
  **Why:**

- How does your policy differ from the policy in Part 3? Why?
  **Answer:**

- How did you implement your policy? e.g., docker cpu-set updates, taskset updates for memcached, pausing/unpausing containers, etc.
  **Answer:**

- Describe the design choices, ideas and trade-offs you took into account while creating your scheduler (if not already mentioned above):
  **Answer:**

3. **[23 points]** Run the following `mcperf` memcached dynamic load trace:

```
$ ./mcperf -s INTERNAL_MEMCACHED_IP --loadonly
$ ./mcperf -s INTERNAL_MEMCACHED_IP -a INTERNAL_AGENT_IP \
           --noload -T 16 -C 4 -D 4 -Q 1000 -c 4 -t 1800 \
           --qps_interval 10 --qps_min 5000 --qps_max 100000 \
           --qps_seed 3274
```

Measure memcached and PARSEC performance when using your scheduling policy to launch workloads and dynamically adjust container resource allocations. Run this workflow 3 separate times. For each run, measure the execution time of each PARSEC job, as well as the latency outputs of memcached. For each PARSEC application, compute the mean and standard deviation of the execution time across three runs. Compute the mean and standard

deviation of the total time to complete all jobs. Fill in the table below. Also, compute the SLO violation ratio for memcached for each of the three runs; the number of data points with 95th percentile latency > 1ms, as a fraction of the total number of datapoints. You should only report the runtime, excluding time spans during which the container is paused.

**Answer:**

| job name | mean time [s] | std [s] |
|---|---|---|
| blackscholes | | |
| canneal | | |
| dedup | | |
| ferret | | |
| freqmine | | |
| radix | | |
| vips | | |
| total time | | |

Include six plots – two plots for each of the three runs – with the following information. Label the plots as 1A, 1B, 2A, 2B, 3A, and 3B where the number indicates the run and the letter indicates the type of plot (A or B), which we describe below. In all plots, time will be on the x-axis and you should annotate the x-axis to indicate which PARSEC benchmark starts executing at which time. If you pause/unpause any workloads as part of your policy, you should also indicate the timestamps at which jobs are paused and unpaused. All the plots will have have two y-axes. The right y-axis will be QPS. For Plots A, the left y-axis will be the 95th percentile latency. For Plots B, the left y-axis will be the number of CPU cores that your controller allocates to memcached. For the plot, use the colors proposed in this template (you can find them in `main.tex`).

**Plots:**

4. **[16 points]** Repeat Part 4 Question 3 with a modified `mcperf` dynamic load trace with a 5 second time interval (`qps_interval`) instead of 10 second time interval. Use the following command:

```
$ ./mcperf -s INTERNAL_MEMCACHED_IP --loadonly
$ ./mcperf -s INTERNAL_MEMCACHED_IP -a INTERNAL_AGENT_IP \
           --noload -T 16 -C 4 -D 4 -Q 1000 -c 4 -t 1800 \
           --qps_interval 5 --qps_min 5000 --qps_max 100000 \
           --qps_seed 3274
```

You do not need to include the plots or table from Question 3 for the 5-second interval. Instead, summarize in 2-3 sentences how your policy performs with the smaller time interval (i.e., higher load variability) compared to the original load trace in Question 3.

**Summary:**

What is the SLO violation ratio for memcached (i.e., the number of datapoints with 95th percentile latency > 1ms, as a fraction of the total number of datapoints) with the 5-second time interval trace?

**Answer:**

What is the smallest `qps_interval` you can use in the load trace that allows your controller to respond fast enough to keep the memcached SLO violation ratio under 3%?

**Answer:**

What is the reasoning behind this specific value? Explain which features of your controller affect the smallest `qps_interval` interval you proposed.

**Answer:**

Use this `qps_interval` in the command above and collect results for three runs. Include the same types of plots (1A, 1B, 2A, 2B, 3A, 3B) and table as in Question 3.

**Plots:**

| job name | mean time [s] | std [s] |
|---|---|---|
| blackscholes | | |
| canneal | | |
| dedup | | |
| ferret | | |
| freqmine | | |
| radix | | |
| vips | | |
| total time | | |