Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Systems@ETH Zürich

Cloud Computing Architecture

Semester project report

Group 022
Ruben Schenk - 18-801-886
Oleh Kuzyk - 21-913-496
Thomas Gence - 18-819-979

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

Instructions

- Please do not modify the template, except for putting your solutions, group number, names and legi-NR.
- Parts 1 and 2 should be answered in maximum six pages (including the questions).
  If you exceed the space, points may be subtracted.

Part 1 [25 points]

Using the instructions provided in the project description, run memcached alone (i.e., no interference), and with each iBench source of interference (cpu, l1d, l1i, l2, llc, membw). For Part 1, you must use the following `mcperf` command, which varies the target QPS from 30000 to 110000 in increments of 5000 (and has a warm-up time of 2 seconds with the addition of `-w 2`):

```
$ ./mcperf -s MEMCACHED_IP --loadonly
$ ./mcperf -s MEMCACHED_IP -a INTERNAL_AGENT_IP  \
           --noload -T 16 -C 4 -D 4 -Q 1000 -c 4 -w 2 -t 5 \
           --scan 30000:110000:5000
```

Repeat the run for each of the 7 configurations (without interference, and the 6 interference types) **at least three times** (three should be sufficient in this case), and collect the performance measurements (i.e., the `client-measure` VM output). Reminder: after you have collected all the measurements, make sure you delete your cluster. Otherwise, you will easily use up the cloud credits. See the project description for instructions how.

(a) **[10 points]** Plot a single line graph with the following stipulations:

- Queries per second (QPS) on the x-axis (the x-axis should range from 0 to 110K). (note: the actual achieved QPS, not the target QPS)
- 95th percentile latency on the y-axis (the y-axis should range from 0 to 8 ms).
- Label your axes.
- 7 lines, one for each configuration. Add a legend.
- State how many runs you averaged across and include error bars at each point in both dimensions.
- The readability of your plot will be part of your grade.

Answer: The single line graph according to the above constraints is shown in figure 1. We averaged 6 runs across. The error bars represent the standard deviation in both directions at each measuring point.

(b) **[6 points]** How is the tail latency and saturation point (the "knee in the curve") of memcached affected by each type of interference? Why? Briefly describe your hypothesis.

Answer: Based on the data we gathered, as seen in figure 1, we can pose the following hypotheses on the impact of each type of interference on the tail latency and saturation point of `memcached`:

- *No interference and L1 data cache interference:* The tail latency remains low compared to the other interferences up until $\sim 95'000$ QPS where the knee in the curve appears. This implies that the L1 data cache has a small impact on the performance of `memcached` and that the system overall can handle requests up to $\sim 95'000$ QPS without any great increase in tail latency. This can be explained with the fact that for a database greater than the size of the L1 data cache and random accesses to that database, the L1 data cache does not have any notable impact on the performance on the operations of `memcached`.

- *Memory bandwidth and last-level cache interference:* The tail latency stays constant up until $\sim 90'000$ QPS where the knee in the curve appears. This indicates that the memory subsystem becomes a bottleneck, compared to the two interferences above, when memory bandwidth or last-level cache interference is present. `memcached` relies on the memory subsystem to store and retrieve data, so interferences in this area can lead to increased latency, which corresponds to the data we collected.

- *L2 cache interference:* Similar to the memory bandwidth and last-level cache interference, the tail latency stays constant up until $\sim 90'000$ QPS where the knee occurrs. However, the latency does not increase by the same amount as in the case of memory bandwidth and last-level cache interference, implying that the L2 cache likely plays a less critical role in `memcached`'s performance. This fits the hypothesis we proposed in the L1 data cache case, where we had even less of an impact in latency. Since the L2 data cache is bigger than the L1 data cache, but smaller than the last-level cache, it makes sense that the impact on the latency is between those two cases.

- *CPU and L1 instruction cache interference:* The tail latency starts comparably high and the knee already occurrs at $\sim 45'000$ QPS. This indicates that the system is heavily bottlenecked by the CPU and L1 instruction cache interference. This can be explained by the fact that both CPU and L1 instruction cache play crucial roles in processing the `memcached` requests. The unexpected drop in tail latency at $\sim 50'000$ QPS might be related to the specific system configuration, workload patterns, or the benchmark's characteristics. It could be that a temporary steady state or a saturation point is reached at $50'000$ QPS, leading to the drop in tail latency.

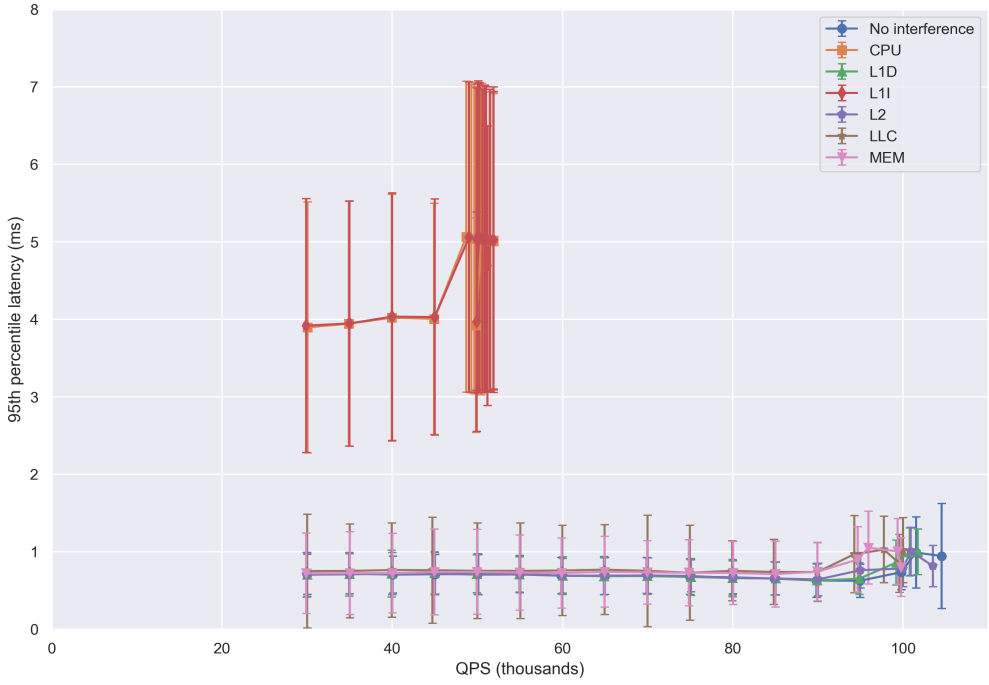

Figure 1: The 95th percentile latency (tail latency) as a function of queries per second (QPS) given different interferences on the server running the `memcached` application. The errorbars shows the standard deviation of the latency at each measuring point.

(c) **[2 points]** Explain the use of the `taskset` command in the container commands for memcached and iBench in the provided scripts. Why do we run some of the iBench benchmarks on the same core as memcached and others on a different core?

Answer: In our experiment, the CPU, L1D, L1i, and L2 benchmarks are running on the same core as the `memcached` application. This makes sense since then they share the same CPU resources, such as the execution units, L1 data cache, L1 instruction cache, and L2 cache. Otherwise, the benchmarks would not impact the same resources as the `memcached` application is using and we could not measure the effective impact of those benchmarks. The LLC and memory bandwidth benchmarks are ran on different cores than the `memcached` benchmark. This makes sense since we want to focus on the impact of shared higher-level resources, such as the last-level cache and the memory bandwidth. Running those benchmarks on the same core as the `memcached` application could lead to interferences on other levels, such as the CPU resources, and we would not be able to safely assign the impact on tail latency to only the LLC or memory bandwidth interference.

(d) **[2 points]** Assuming a service level objective (SLO) for memcached of up to 1.5 ms 95th percentile latency at 65K QPS, which iBench source of interference can safely be collocated with memcached without violating this SLO? Briefly explain your reasoning.

Answer: According to the data we collected, it should be safe to collocate the L1D and L2 interferences with `memcached` while still respecting the SLO. The tail latency up until 65K QPS is below 1.5ms and the standard deviations are comparably low, indicating relatively consistent performance. LLC and memory bandwidth interferences could also be safely collocated according to our data while still respecting the SLO since, again, the tail latencies are below 1.5ms. However, both interferences pose slightly higher standard deviations, which could indicate a slightly higher risk of occasional SLO violations. For CPU and L1I interferences, the combination of both high tail latencies (over 1.5ms) and the high variability, it is not possible to collocate those while respecting the SLO of 1.5ms.

(e) **[5 points]** In the lectures you have seen queueing theory. Is the project experiment above an open system or a closed system? What is the number of clients in the system? Sketch a diagram of the queueing system and provide an expression for the average response time. Explain each term in the response time expression.

Answer: In our case the system is a *open system.* Although we have a limited number of clients, those clients can generate load even if their previous requests have not been serviced yet. The system we set up has *16 clients.* There is one client virtual machine on which we create 16 threads in the mcperf client load agent, each acting as one client. The corresponding queuing system is modelled in figure 2. Assuming that both the arrival process and the service times are exponentially distributed, the arrival process with parameter λ and the service times with parameter μ, we can model the system as a M/M/1 system (single server, infinite buffer, FCFS queue). As seen in the lecture, the average response time R in a M/M/1 system is given by:

$$R = \mathbf{E}[w] = \frac{1/\mu}{1-\rho} = \frac{1/\mu}{1-(\lambda/\mu)} = \frac{1}{\mu-\lambda} \tag{1}$$

In equation 1, R is the average response time, μ is the service rate, and λ the arrival rate. Note that the system must be stable for this equation to hold, i.e. it must be true that $\lambda < \mu$.

Part 2 [30 points]

1. **Interference behavior [19 points]**

   (a) **[11 points]** Fill in the following table with the normalized execution time of each batch job with each source of interference. The execution time should be normalized to the job's execution time with no interference. Round the normalized execution time to 2 decimal places. Color-code each field in the table as follows: **green** if the normalized execution time is less than or equal to 1.3, **orange** if the normalized execution time is over 1.3 and up to 2, and **red** if the normalized execution time is greater than 2. Briefly summarize in a paragraph the resource interference sensitivity of each batch job.

Workload	none	cpu	l1d	l1i	l2	llc	memBW
blackscholes	1.00	1.31	1.24	1.55	1.23	1.45	1.34
canneal	1.00	1.17	1.16	1.36	1.24	1.34	1.29
dedup	1.00	1.67	1.25	2.27	1.25	2.19	1.59
ferret	1.00	1.90	1.11	2.43	1.02	2.52	1.98
freqmine	1.00	1.99	1.05	2.02	1.01	1.79	1.56
radix	1.00	1.05	1.11	1.14	1.10	1.54	1.07
vips	1.00	1.71	1.60	1.97	1.57	1.94	1.60

   Answer: BLACKSCHOLES is moderately sensitive to interference, with the highest sensitivity to L1I and LLC interference, and the least sensitivity to L1D and L2 interference. CANNEAL behaves similarly to BLACKSCHOLES, but is even less sensitive to CPU and memBW interference. DEDUP is highly sensitive to L1I and LLC interference and moderately sensitive to CPU and memBW interference. FERRET behaves similarly to DEDUP, showing even stronger sensitivity to L1I, LLC, CPU, and memBW interference. FREQMINE is highly sensitive to CPU and L1I interference and moderately sensitive to LLC and memBW interference. RADIX exhibits the lowest overall sensitivity to interference among the other workloads, only showing moderate sensitivity to LLC interference. VIPS is highly sensitive to L1I and LLC interference and moderately sensitive to CPU, L1D, L2, and memBW interference.

   In summary, most batch jobs show the highest sensitivity to L1I and LLC interference, and the least sensitivity to L1D and L2 interference. DEDUP, FERRET, and FREQMINE are the most sensitive workloads and RADIX is the least sensitive workload according to the data we collected.

   (b) **[8 points]** Explain what the interference profile table tells you about the resource requirements for each application. Which jobs (if any) seem like good candidates to collocate with memcached from Part 1, without violating the SLO of 2 ms P95 latency at 40K QPS?

   Answer: Taking the sensitivities into account we elaborated on in the previous task, we can argue about the resource requirements for each application as follows:

   - BLACKSCHOLES: Relies on a mix of resources, with higher reliance on L1I and LLC.
   - CANNEAL: Relies more on L1I and LLC and less on CPU, L1D, L2, and memBW.
   - DEDUP & FERRET: Relies heavily on L1I and LLC resources and moderately on CPU and memBW.

- FREQMINE: Moderate reliance on memBW and LLC resources and heavy reliance on CPU and L1I resources.
- RADIX: Moderate reliance on LLC and less reliant on other resources.
- VIPS: Highly reliant on L1I and LLC resources, moderately reliant on all other resources.

As we can see in part 1, `memcached` has the worst performance when running together with L1I or CPU interference. Thus, it is crucial to collocate it with jobs that are not reliant on L1I or CPU resources as well. Therefore, DEDUP, FERRET, and FREQMINE are definitely not candidates for collocation. VIPS shows less reliance on CPU and L1I resources than the previously mentioned workloads, but still enough to violate the SLO of 2ms if collocated with `memcached`. Similarly, BLACKSCHOLES is even less reliant on those resources, but still the L1I-reliance could cause problems on the SLO. In conclusion, the two candidates for collocation with `memcached` are CANNEAL and RADIX since both show low reliance on CPU and L1I and their normalized execution times together with the tail latency of `memcached` at 40K QPS seem to respect the SLO of 2ms.

2. **Parallel behavior [11 points]**

Plot a single line graph with speedup as the y-axis (normalized time to the single thread config, Time_1 / Time_n) vs. number of threads on the x-axis (1, 2, 4 and 8 threads, see the project description for more details). Briefly discuss the scalability of each application: e.g., linear/sub-linear/super-linear. Do the applications gain significant speedup with the increased number of threads? Explain what you consider to be "significant".

Answer: Figure 3 shows the scalability curve of each application.

- BLACKSCHOLES: We can see that the scalability is *sub-linear* and plateaus after 4 threads, where the speedup becomes constant.
- CANNEAL: The speedup is increasing with the number of threads but not proportionally, the scalability is thus *sub-linear.*
- FERRET: The speedup is significant from 1 to 2 threads but diminishes after that. The scalability is *sub-linear.*
- FREQMINE: Similar scalability to that of BLACKSCHOLES with *sub-linear* scalability and plateauing speedup after 4 threads.
- DEDUP: *Sub-linear* scalability until 4 threads but with a significant drop when moving from 4 to 8 threads, suggesting potential bottlenecks or issues with parallelization.
- RADIX: Nearly linear (but still *sub-linear*) speedup until 4 threads, clearly sub-linear scalability after 4 threads.
- VIPS: Similar scalability to RADIX with *sub-linear* speedup as the it does not grow proportionally with the number of threads.

In conclusion, it can be observed that the scalability for each application results in a sub-linear speedup. We might argue that, although no application exhibits ideal linear speedup, RADIX and VIPS demonstrate a significant speedup compared to the other applications, as they consistently improve their performance with additional threads.

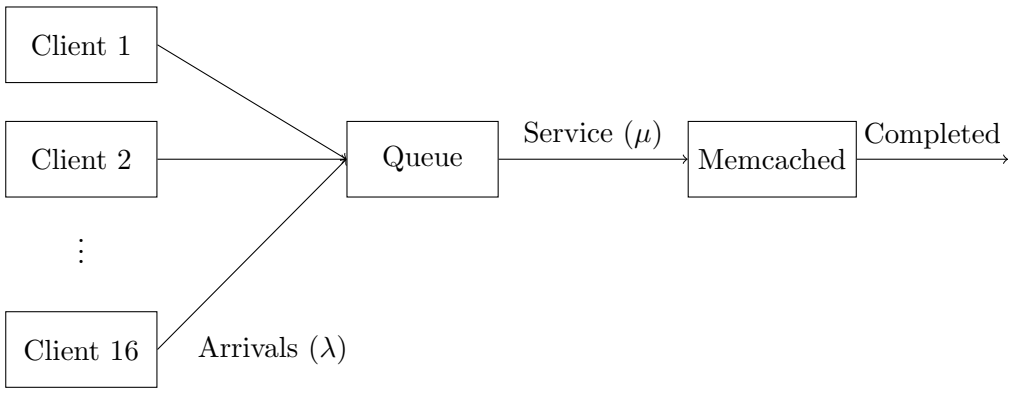

Figure 2: Diagram of the queueing system for the `memcached` application with 16 clients.

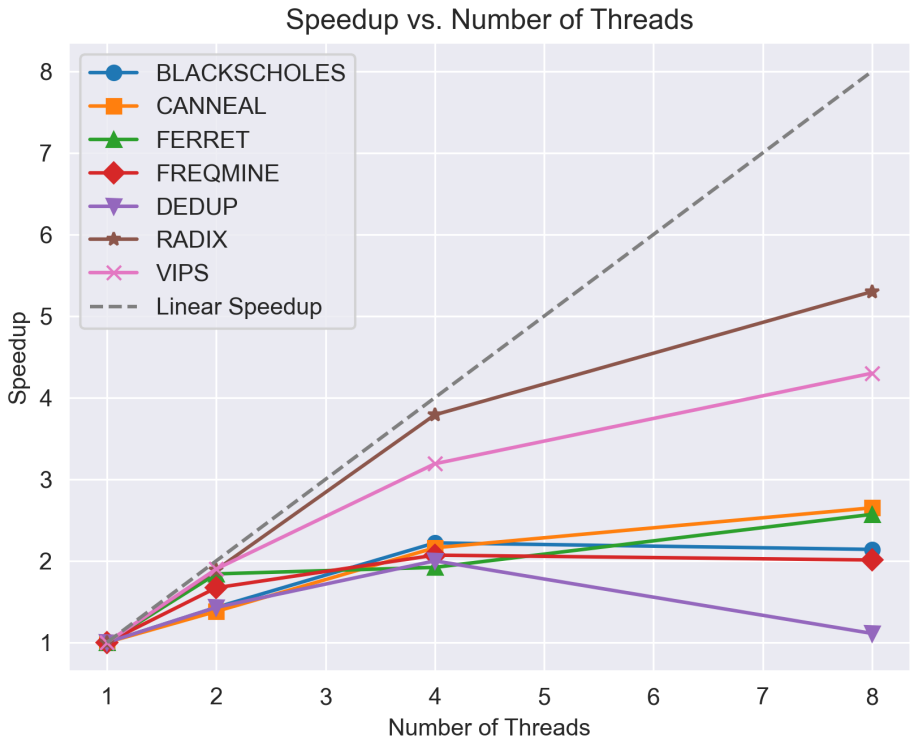

Figure 3: The speedup achieved of 7 different applications when running on 1, 2, 4, and 8 threads. The dashed line represents ideal linear speedup.