Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Systems@ETH zürich

Cloud computing architecture

Semester project report

Group 027
Jiale Chen - 20-961-504
Ran Liao - 20-949-186
Xintian Yuan - 20-951-778

Systems Group
Department of Computer Science
ETH Zurich
April 12, 2021

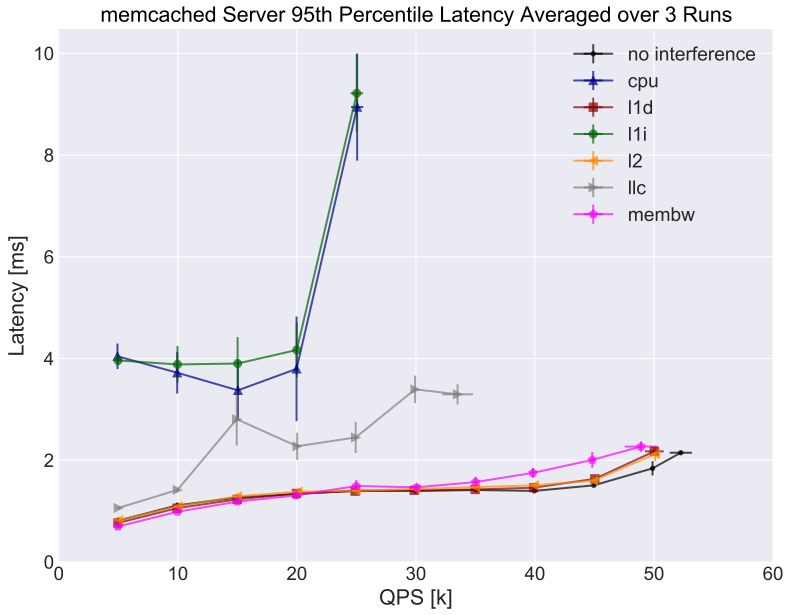
Instructions

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

Part 1 [20 points]

1. **[10 points]** Plot a single line graph with 95th percentile latency on the y-axis (the y-axis should range from 0 to 10 ms) and QPS on the x-axis (the x-axis should range from 0 to 55K). Label your axes. State how many runs you averaged across (we recommend three) and include error bars. There should be 7 lines in total in your plot, showing the performance of memcached running with no interference and six different sources of interference: `cpu`, `l1d`, `l1i`, `l2`, `llc`, `memBW`. The readability of your plot will be part of your grade.

   Solution: Due to resource contention introduced by the interference, the target QPS cannot be achieved after 25K for `cpu` and `l1i` and after 35K for `llc`, thus the points are overlapped. To ensure readability, we averaged the overlapping points to a single point.



2. **[6 points]** Describe how the tail latency and saturation point (the "knee in the curve") of memcached is affected by each type of interference. Also describe your hypothesis for why performance is affected in this way.

   Solution: The `l2`, `l1d` and `memBW` interference have minor influences on tail latency. Their lines with interference are quite similar to the line without interference. Their saturation points are around 30K QPS. `llc` interference affects tail latency more significantly when QPS is larger than 10K. `cpu` and `l1i` interference increase tail latency significantly from less than 2 ms to more than 4 ms. The saturation point is around 20K QPS. In my opinion, memcached is a CPU intensive job and requires a large amount of CPU resources. Therefore, interference on `cpu` and `l1i` cache makes memcached very inefficient. The data accessed in memcached might be quite random and have little locality, thus, tail latency is largely dominated by the speed of moving data between memory and CPU. The cache is somewhat irreverent and has little influence.

3. **[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?

Solution: Each core has its own separate L1 and L2 cache, therefore, interference on such caches must be conducted on the same core. Since the last level cache and main memory are usually shared across all cores, it's fine to do such interference on a different core. And CPU interference should affect the target core directly.

4. **[2 point]** Assuming a service level objective (SLO) for memcached of up to 2 ms 95th percentile latency at 40K QPS, which iBench source of interference can safely be collocated with memcached without violating this SLO? Explain your reasoning.

   Solution: `l1d`, `l2`, and `memBW`. According to the figure in the previous question, only these 3 kinds of interference have minor influence on tail latency, making it less than 2 ms on 40K QPS. Other interference will make memcached unable to reach 40K QPS at all.

Part 2 [25 points]

1. **[12 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. 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. Summarize in a paragraph the resource interference sensitivity of each batch job.

   Solution: The readings are taken from the "real" time for `dedup`, `blackscholes` and `canneal` as indicated in the logs. For `ferret`, `freqmine` and `fft` benchmarks we used "QUERY TIME", "FPgrowth cost" and "Computation Time" respectively for the calculations. To get the values in the table, we first averaged the readings from three repeated trials, then normalized the ones with interference respect to the normal ones. The results are summarized in the following table.

Workload	`none`	`cpu`	`l1d`	`l1i`	`l2`	`llc`	`memBW`
dedup	1.0	1.52	1.24	2.21	1.24	2.13	1.66
blackscholes	1.0	1.42	1.42	1.89	1.38	1.73	1.45
ferret	1.0	1.98	1.06	2.77	1.05	2.78	2.20
freqmine	1.0	2.04	1.01	2.06	1.02	1.92	1.61
canneal	1.0	1.40	1.46	1.74	1.44	2.20	1.61
fft	1.0	1.33	1.33	2.07	1.35	2.12	1.59

   As clearly indicated in the table, `dedup` is most sensitive to `l1i` and `llc` interference while not quite affected by `l1d` and `l2` interference. The `ferret` batch has similar behaviors but higher sensitivity with `memBW` interference. `blackscholes` is moderately influenced by all interference. `freqmine` is more sensitive to `cpu` and `l1i`, but least sensitive to `l1d` and `l2` interference. It was also affected by `llc` and `memBW` to some extent. The `canneal` batch is only primarily influenced by `llc` and the `fft` batch is largely affected by `l1i` and `llc`. All other interference had fair but limited influences on `canneal` and `fft` batches.

2. **[3 points]** Explain in a few sentences what the interference profile table tells you about the resource requirements for each application. Which jobs (if any) seem like good candidates to

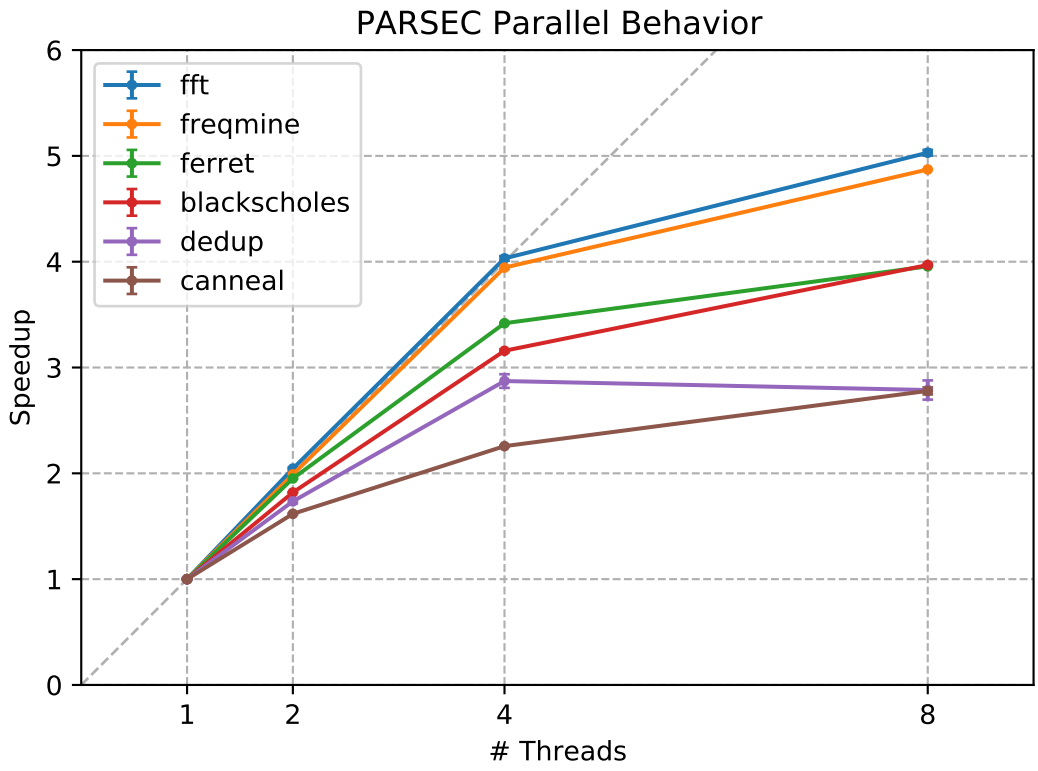
collocate with memcached from Part 1, without violating the SLO of 2 ms P95 latency at 40K QPS?

Solution: The more an application is affected by certain type of interference means the more resources is needed for the application in this category. For example, `dedup` needs more `l1i` and `llc` caches. `blackscholes` needs comparatively amount of each type of resource. `ferret` requires large `l1i` and `llc` caches and significant amount of memory. `freqmine` requires more CPU usage and instruction caches. `canneal` demands mostly the `llc` cache. `fft` demands `l1i` and `llc` cache. Last but not least, `dedup`, `ferret` and `freqmine` require very small amount of `l1d` and `l2` caches.

`blackscholes` is a good candidate to collocate with memcached since it is not affected by `cpu`, `l1i` or `llc` interference significantly.

3. **[10 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. Briefly discuss the scalability of each application: e.g., linear/sub-linear/super-linear. Do any of the applications gain a significant speedup with more threads? Explain what you consider to be "significant".

   Solution: `fft` and `freqmine` are linear with 2 or 4 threads, but sub-linear with 8 threads. `ferret` is linear with 2 threads but sub-linear with more than 2 threads. `blackscholes`, `dedup`, and `canneal` are all sub-linear with multiple threads. `dedup` even has a slight performance drop with 8 threads compared to that with 4 threads. We consider a speedup to be "significant" if it is larger than 3. Thus, `fft`, `freqmine`, `ferret`, and `blackscholes` gain a significant speedup with at least 4 threads. `canneal` might gain a significant speedup with 16 threads if the machine has 16 cores. `dedup` is unlikely to gain a significant speedup with any number of threads.



Note: Each experiment was repeated 3 times. We draw the points using the average values and the error bar using the standard deviations. We used the "real" time for `blackscholes`, `canneal`, and `dedup`. We used the "QUERY TIME" for `ferret`. We used the "Computation Time" for `fft`. We used the "FPgrowth cost" for `freqmine`.