Project #3: Profiling IO Path

In our 3rd and last mini-project, our goal is to (1) get familiar with writing scripts for profiling, (2) understanding the io_path of linux kernel, and (3) put your CSP knowledge into further practice.

The profiling tool we will be using is perf, and we will use fio tool to create the workloads necessary to profile the IO path.

As the hardware platform, you can use the google cloud resources as usual for this project. I recommend you to configure an instance with an SSD.

Both perf and fio should be installed in the compute instance you are using, but if not, you can install them using the corresponding package manager.

You have till the end of semester to complete this project.

Getting started with fio:

Familiarize yourselves with fio. https://fio.readthedocs.io/en/latest/fio_doc.html has the list of commands and explanations.

For example, the following line sends 4KB random read IO requests for a minute that covers a region of 1GB with io-depth of 1 and direct IO (no OS page cache in action) using io_uring (see slides from week 6's SSD overview). The disk being used would be the disk holding the directory you invoke this command at.

$ fio --name=randread --ioengine=io_uring --iodepth=1 --rw=randread --bs=4k --direct=1 --size=1GB --runtime=60

Try to understand these parameters and output better, change the parameters, observe what happens.

Getting started with perf:

Familiarize yourselves with perf.

We will use perf in two modes: (1) to observe what happens on the IO path when we issue a particular type of IO, (2) to measure the software events invoked for different types of IOs.

For #1; we will use "perf record" to record the profiling info and FlameGraphs to visualize the output. Follow the examples at https://www.brendangregg.com/perf.html#FlameGraphs for this.

For #2; we will use "perf stat -e <events to keep track>". Follow examples at https://www.brendangregg.com/perf.html#Events for this. If you invoke "perf list", you will see the list of software and hardware events you can trace.

When you run perf for the first time, it may give you some warnings. This is because, to be able to perform detailed profiling of kernel events, one needs to resolve the program symbols at that level, which isn't allowed by default for security reasons. Similarly access to tracing low-level events like context-switches or hardware events are restricted by default. In this project, you should specifically rewrite

"/proc/sys/kernel/perf_event_paranoid" to -1 and "/proc/sys/kernel/kptr_restrict" to 0. Please google these two variables and learn what they are.

Project goal:

You will profile the workloads you create with fio using perf in the two modes described above.

In this project, I want you to specifically get results by changing the ioengine parameter to sync, libaio, io_uring. In addition, in the case of io_uring, you will experiment with turning polling option on/off. You can find the option to turn this on/off in the fio documentation. Keep direct IO on, and by default you can just focus on random reads.

Part 1: Firstly, you profile where does time go in the IO path and see how the IO path looks like in parallel? You will use *perf record* and visualize your results using flamegraphs. (Option #1 under *Getting started with perf*). Flamegraphs can be exported in svg format, which you can include as appendix in your report. This format also allows you to dig deeper into a specific part of the profiling results (click on the function you are interested in on svg and see).

In this part, you will need just four experiments: sync, libaio, io-uring with/without submission polling enabled.

Part 2: Then, you collect the following metrics.

From fio, get throughput (IOPs and MB/s) and latency (average and 99%) results.

From *perf stat*, focus on the metrics of context switches, page faults, and migrations per IO.

You can collect the numbers from both tools as you are running them together.

In addition to the parameters mentioned above, for this part, pick a parameter (workload, num-jobs, io-depth, block size, etc.) that you want to experiment further with and have experiments with 4 different values for it. For example, you can choose to observe the impact of io-depth and experiment with io-depths of 1, 2, 4, and 8.

Furthermore, create experiments in two modes here: (1) where perf is attached to fio from the beginning and (2) where perf is attached to fio for a duration that excludes the beginning and the end of the fio run (i.e., fio runs for a minute, but perf traces for 20secs or 30secs of this run). For the latter case, I recommend you to write a bash script to handle this.

Some questions to answer in parallel:

(1) Do you see the same output when you invoke "perf list" on both google compute instance and your own laptop? If not, what is the difference and why is there a difference? If yes, did you expect this and why?
(2) Are you able to trace hardware events with perf in the google cloud platform? If yes/no, why?
(3) What are the pros/cons of collecting events with perf in the two experimental modes described above?

The hand-in (part of the exam hand-in)

(1) The scripts / commands that you used to run the experiments.

(2) A report that consists of the following sections, which should be at most ~4 pages (with minimum 11pt font size – excluding the flamegraphs)

a) *Experimental methodology and setup.* Here you explain your experimental methodology and setup. What is the system? What is the workload? What are the metrics? What are the parameters? What are the tools we are using? How many times you ran the experiments? … In addition, add a limitations and assumptions subsection, where you answer the two questions above.
b) *Interpretation of results.* Here you report the results and interpret them. Please be clear and precise. For example, if you have graphs, the x-axis, y-axis, graph legend should have clear titles and should be explained. In addition, you should have a brief overview of the numerical trends for each experiment relating these trends to your metrics. For example, does the three io-engines behave the same? If yes, was this expected? If not, why?
c) *Conclusion.* Here include reflections for overall project and what you have learned.

Please do not hesitate to ask if you have any questions!