CS-471 Stage 2 Project report

Souleyman Boudouh, Clément Charmillot, Emilien Guandalino

1 Recall from last time

We shortly remind here the previous status of the project at the end of the stage 1 report.

Problem: We are trying to create a load-aware load generator which can maximize the load/throughput for a given workload within a tail latency constraint.

Hypothesis: When increasing the load, the increase in latency can either be inherent to the given workload, ie *inherent latency*, or simply caused by the change itself, ie *transient latency*. We expect the transient latency to settle down after some time.

Solution: Factor analysis, ie model the latency as a function of throughput and different factors which we manually classify as either inherent or transient. Maximum is reached when inherent latency reaches tail latency constraint.

2 Updated Research Problem

Hypothesis: We continue our attempt to solve this problem by maintaining the same hypothesis, which we believe to be true. However we have completely changed our proposed solution and abandoned the factor analysis. Here are a few reasons why we have decided to change:

1. Factor analysis is a much harder problem than what is necessary for our purpose. It requires understanding exactly from where the latency is coming from, proper modeling and precise measurement of each factor during execution. This is an entirely new research problem, and although interesting, it is beyond the scope of our project.

2. It requires the researcher to implement measurement and adaptation of all factors for each new workload which we are testing. This is a lot of work, which defeats the initial purpose of simplifying the testing process.

3. We can achieve the same result with a simpler approach, which detail below.

New Solution: Based on our stated hypothesis, we already assume that the variance goes through a transient phase until it eventually converges to its inherent value. We can therefore detect the point at which the transient latency has settled down by experimentally looking at the convergence rate of the latency. Figure 1 shows what this behavior would look like, and we expect to find similar effects in latency measurements.

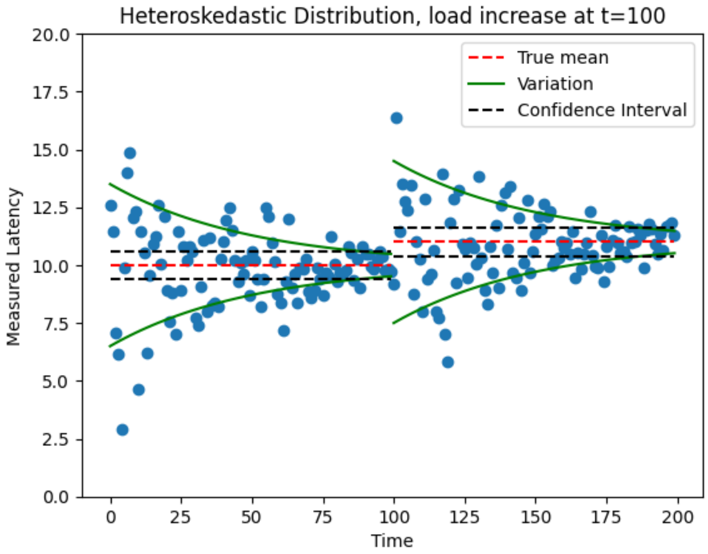

Figure 1: Hypothetic Latency Distribution

We can see on the graph that with time, variance decreases as transient latency settles down. At time $t = 100$, we have reached our confidence interval, at which point we decide that it is safe to increase the workload. This change causes the variance to increase again, before settling down to a new maximum. We iterate through this process, until we have good statistical certainty that we have reached maximum throughput within a tail latency constraint.
Compared to factor analysis, this strategy only requires us to experimentally measure latency from our system which we treat as a black box. This however raises some new theoretical questions which we explain in the next section.

Related Work: As explained in the Stage 1 report, and after further investigation, we find a large amount of literature around tail latency, its characterization and its sources. Once again, our purpose is different from those papers since we are in a testing environment in which we have control over the system load, unlike in production.

3 Questions, Hypotheses and Test

In this section, we discuss the new theoretical questions which are raised by our new approach. In the next section, we will give more information as to where each of those questions fit into the global picture.

Questions: Here are the new theoretical issues which we are currently working on:

1. *Sampling Interval*: When evaluating a workload, we perform a set number of operations over which we then measure latency for this period. However, there are no simple solutions to determine how long this sampling period should be in order to guarantee that our measurement is accurate enough. There are multiple parameters which we should take into account to determine this window:

(a) Accuracy & Confidence Intervals: How accurately do we want to estimate latency? How much certainty do we want about our measure ? More accuracy and certainty imply larger sampling windows.

(b) Variability: How variable is the metric that we are observing ? Greater variability implies more samples for accurate estimation.

(c) Effect Size: How large is the effect size that we are measuring ? Accurately measuring small changes require more samples.

(d) Time Homogeneity: How to correctly sample a distribution which changes over time ? This will affect the different sampling windows that we consider for measurement.

We are still looking into different statistical tools, such as Power Analysis, which enable us determine sampling size when considering all those factors. Since we still have not yet figured out a full solution, we have used a fixed sampling window size when performing our measurements.

2. *Convergence Detection*: There are two levels at which we need to detect convergence:

   2.1. At each iteration: When do we decide that we have converged to inherent latency? If we are above the tail latency requirement we may have to decrease load again, but we must first be sure that we have actually converged.

   2.2. Across iterations: When do we decide that we have found a global maximum and output a result ? We may oscillate around the optimum for many iterations.

3. *Workload Increase Step Size*: When we decide to increase workload, by how much should we increase it ?

4. *Hysteresis*: When evaluating complex systems, even with proper methodology and the correct amount of samples, two different runs might converge to different values. This is a difficult problem to address, and we will first try to optimize for a single run, before trying to optimize across runs.

Testing: Concerning the test of our hypothesis, we will give more details in the following section. For the theoretical issues mentioned above, we first need to ensure that our mathematical derivations are rigorous, then we need test our code implementations and finally we can run the workloads and experimentally tune the parameters, ex: confidence interval size, for best results. As a reference, we can compare the output of our solution with the results given in the Tailbench paper.

4 Progress, Result and Analysis

Experimental Setup: In order to test our hypothesis, we used the code provided by the Tailbench testing suite. For reasons detailed in the ”Challenges” section, we decided to use Silo, an OLTP database, with a TPC-C benchmark. We ran our experiment in integrated configuration, i.e. all operations happen in memory (no disk, no network access), with a single thread and a single warehouse.

Hypothesis test: Here are the results that we obtained:

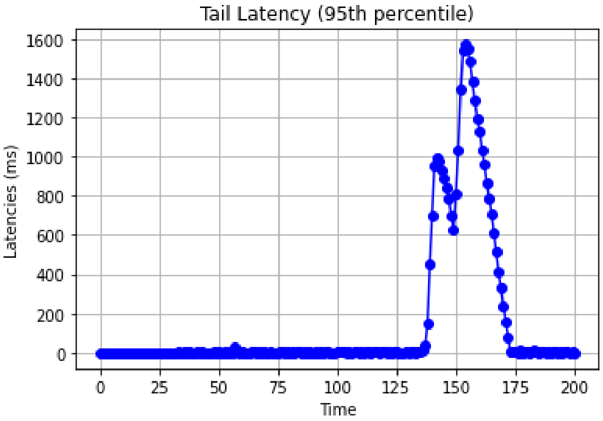

Figure 2: Tail Latency Measurement

As we can see, the tail latency greatly increases when we perform a change in workload, and eventually converges back to its inherent state. Now that we have a good reason to believe that our hypothesis is correct, we propose an algorithm which exploits this hypothesis in order to solve our problem.

Algorithm Design: We now give the algorithm that we have designed in order to maximize throughput within tail latency constraints. The numbers on the right indicate to which theoretical problem each method corresponds to.

Algorithm 1 Load-Aware Load Generator

```
Initialize workload data structures
running ← true
while running do
    num_requests ← find_sampling_window()                                  Problem 1.
    run_workload(num_requests)
    latency ← aggregate_latencies()
    converged ← latency_convergence(latency, confidence_interval)         Problem 2.1
    if converged then
        if global_convergence(latency, latency_requirement) then          Problem 2.2
            return (load, latency)
        else if latency < latency_requirement then
            increase_load()                                                Problem 3.
        else if latency > latency_requirement then
            decrease_load()                                                Problem 3.
        end if
    end if
end while
```

In our current implementation, we have simplified each sub-problem in order to be able to make first measurements for our hypothesis test. Improving this algorithm will be the next focus of our project.

5 Challenges

We encountered many unexpected practical challenges when trying to verify our hypothesis. The first one is working with the Tailbench repository code. The Tailbench suite uses Harness, a tail latency measurement software, to run multiple different benchmarks which have been adapted for this purpose. Both the Harness code and the benchmarks have not been maintained for years, and come with multiple dependency problems, various compilation errors and segmentation faults which we had to debug with GDB. Understanding and making changes to such a large codebase was tedious and time-consuming.

Another challenge that we had was to adapt the way Tailbench performs latency measurements. Tailbench is designed to aggregate statistics about latencies during execution and then dump them into a binary file which we have to parse with a python script afterwards. This is unsuitable for our purpose, since we have to take decisions based on the latency during the execution. If we are not able to modify the workload and observe latency in live, then all of our analysis falls apart. We therefore had to modify the code in order to continuously iterate over a given workload and collect the associated latencies.

6 Plan

We do not have any team member changes. In terms of priority, now that we have verified our hypothesis, we can move on to the next step which means: finishing our algorithm implementation, experimenting with different solutions for the theoretical problems, and evaluating them on our benchmark. Our schedule now reflects those objectives.

7 TLDR

Research: We have not changed our goal or hypothesis. We have changed our approach. Now we treat our system as a blackbox and only measure latency.

Progress: We have tested our initial hypothesis, it works. We have implemented a simple algorithm based on that. A few theoretical questions remain to improve the algorithm.

Next steps: Implement different solutions to the problem, evaluate them.