Measuring Latency: Am I doing it right?

Marios Kogias *[1], Christos Kozyrakis[2] and Edouard Bugnion[1]

[1]EPFL
Email: {marios.kogias, edouard.bugnion}@epfl.ch

[2]Stanford University
Email: kozyraki@stanford.edu

1 Introduction

The ever increasing hardware speeds has led systems and application designers to push software to its limits and create applications with microsecond response times. Moreover, such interactive applications, e.g. websearch, run at massive scales, with large fan-in and fan-out patterns, where the importance of tail-latency becomes crucial to guarantee Service Level Agreements (SLAs). So being able to measure latency accurately is vital to debug those system, identify their behaviour and specify proper SLAs.

Accurate latency measurement proves to be very challenging. Unlike throughput, latency is a very sensitive metric that is affected by multiple factors, both within the application and the operating system [2], but also the measuring client itself. Consequently, we are faced with two major challenges related to latency measurement. We need tools that are able to measure latency in such a low microsecond scale, while independently of the tools, we need an accurate an sound methodology that is able to produce unbiased and realistic results that are collected under settings that highly resemble a production environment.

In this work we first collect and analyze some common methodology pitfalls that have been overlooked by previous approaches [9] and then compare existing tools and identify missing features, related to the experiment methodology, that can either lead to more statistically sound results or reduce necessary resources (time and compute) for latency experiments.

2 Methodology Pitfalls

Specifying the system workload correctly is important to run latency measuring experiments. Apart from the application specific characteristics, e.g. key-value size and popularity distributions for key-value stores, there are generic workload characteristics that depend on the way the system is deployed and its interaction with clients. Specifically, we focus on the number of connections, the open vs closed system model and the distributions that characterize the workload.

The number of connections that a system is expected to serve is a crucial part of the workload definition. Despite the guidelines for some of the existing measuring tools that suggest adding more clients for a higher load, the workload itself is independent from the load level. Different connections might be treated differently given NIC features such as RSS and Flow Director, while the system might behave differently under different number of connections, e.g. connection scalability issues. Moreover, the workload itself might be inherently imbalanced across connections. This should be reflected in the latency experiment and the tools should capture such cases.

Each system that serves incoming requests can be modeled as a closed or an open loop system. Open loop systems resemble more systems with a larger number of connections, where the incoming requests are characterized by an inter-arrival distribution. This inter-arrival distribution severely affects latency measurement given that it can lead to queuing effects, even in low loads, in cases of bursty traffic. In closed loop systems there is one pending request per connection and accepting the response for a request allows sending the next one. Although there is no inter-arrival distribution, closed loop systems are characterized by the distribution of the *think time*, the time between accepting a response and sending the next request, that can lead to different latency results. Both models exist in datacenter applications and even the same application can be deployed in both open and closed loop setups. Misidentifying the system setup [8] or violating workload distributions leads to inaccurate measurements.

Although latency experiments measure round-trip time and don't differentiate between service and queuing time, it is crucial to understand what are the main latency contributors, which change under different load levels, to enable targeted optimizations. At low-loads the main contributor

*student

Feature/Tool	memslap	YCSB	Cloudsuite	Mutilate	Treadmill
Distributed Coordination	No	No	No	Yes	No
Measuring Model	Symmetric	Symmetric	Symmetric	Asymmetric	Symmetric
Dynamic Histograms	No	Yes	No	No	Yes

Table 1: Measuring Tools Comparison

is the server service time, while at higher loads queuing effects dominate latency. According to the central limit theorem for heavy traffic queueing systems [4] the waiting time in any G/G/m system under heavy load converges to an exponential distribution. Consequently, any system optimizations that change the distribution of the service time will not be visible in round-trip latency measurements under heavy load because they are amortized by the queuing effects. Thus, latency versus throughput graphs, instead of single load-point measurements, are necessary to understand the system behaviour.

3 Missing Features

Table 1 compares existing latency measurement tool features that are related to methodology questions. We focus on Mutilate [5], YCSB [1], Treadmill [9], memaslap [6] and CloudSuite [3] and features such as distributed coordination since multiple machines are necessary to achieve connection scaling and certain load levels, the measuring model (asymmetric vs asymmetric), namely a classification of clients among latency-measuring and load-generating, or client equality, and dynamic histograms to avoid mistakes from statistical aggregation of results.

However, apart from those we argue that there are more features related to methodology that can be integrated in latency measuring tools to achieve better quality of results. Specifically, we focus on **automated experiment duration adjustment**, **identification of inter-arrival distribution violations** due to client load and **heavy-tailed distribution identification**.

Existing tools require the user to manually specify the experiment duration and the duration of warm-up and cool-down phases, if any. The last two phases exist either to avoid errors coming from the experiment setup, e.g. launching loading threads, or to bring the system to a stable state in order to measure it. Manually selecting these durations can lead to unnecessarily running the experiment longer, while the system has already converged or stopping the experiment and reporting results even that have not converged yet. Instead, those values can be identified by the measuring tool based on a control loop that leverages online statistical processing of the collected results, such as confidence intervals. A similar control loop could be used to identify the server load level, so that the latency vs throughput experiment moves faster at low loads while it's more detailed when the system is close to saturation.

Having discussed the importance of the inter-arrival distributions, latency measuring tools should be able to identify whether the client respects the inter-arrival distribution and should stop the experiment in case of violations. This would eliminate the effects of client induced latency and deployment setup misrepresentation.

Finally, measuring latency for services with heavy-tailed distribution poses extra challenges. Heavy tailed distributions have an infinite variance and as a result the Central Limit Theorem can not be applied. Consequently, no analysis based on confidence intervals can be performed. Moreover, statistical aggregation over heavy-tailed distributions is prone to statistical error, no matter how big is the sample size. Outliers that unpredictably appear can significantly affect the result and prevent convergence even after multiple executions of the same experiment. Thus, latency measuring tools should be able to identify whether the collected results come from a heavy tailed distribution [7] and if they do, warn about their validity else proceed with performing a confidence analysis.

References

[1] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. Benchmarking cloud serving systems with YCSB. In *SOCC* (2010), pp. 143–154.

[2] Dean, J., and Barroso, L. A. The tail at scale. *Commun. ACM 56*, 2 (2013), 74–80.

[3] Ferdman, M., Adileh, A., Koçberber, Y. O., Volos, S., Alisafaee, M., Jevdjic, D., Kaynak, C., Popescu, A. D., Ailamaki, A., and Falsafi, B. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *ASPLOS-XVII* (2012), pp. 37–48.

[4] Kingman, J. F. C., and Atiyah, M. F. The single server queue in heavy traffic. In *Cambridge Philosophical Society 57* (1961), pp. 902–904.

[5] Leverich, J., and Kozyrakis, C. Reconciling high server utilization and sub-millisecond quality-of-service. In *EUROSYS* (2014), pp. 4:1–4:14.

[6] memaslap. http://docs.libmemcached.org/bin/memaslap.html.

[7] Nair, J., Wierman, A., and Zwart, B. The fundamentals of heavy-tails: properties, emergence, and identification. In *SIGMETRICS* (2013), pp. 387–388.

[8] Schroeder, B., Wierman, A., and Harchol-Balter, M. Open Versus Closed: A Cautionary Tale. In *NSDI* (2006).

[9] Zhang, Y., Meisner, D., Mars, J., and Tang, L. Treadmill: Attributing the Source of Tail Latency through Precise Load Testing and Statistical Inference. In *ISCA* (2016), pp. 456–468.

Measuring Latency: Am I doing it right?

Marios Kogias [1], Christos Kozyrakis [2], Edouard Bugnion [1]

1 ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

2 Stanford University

Motivation

Measure RPC/KV-store latency:
- Accurately
- Efficiently

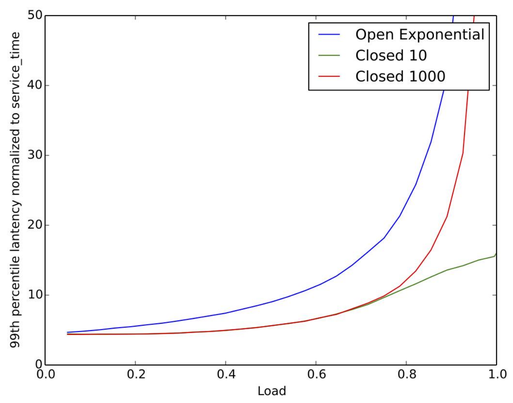
Problems:
- Many tools
- Incompatible results
- Tools bound to methodology

Decouple:
1. **Workload**: How the application is deployed and run?
2. **Methodology**: How the latency measurements are collected and processed?
3. **Tool**: Which software is used for the experiment?

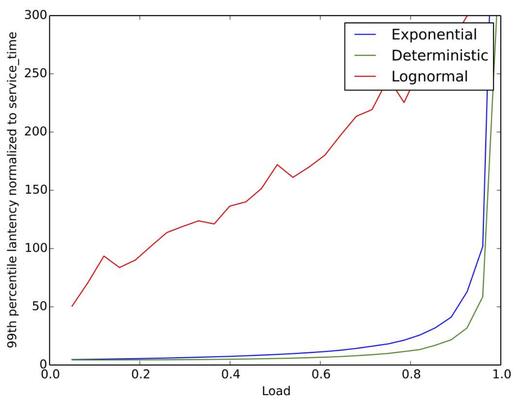
Workload

Workload is defined by:
- Protocol Used (e.g Memcached)
- Number of connections
- System Model (Open vs Closed loop)
- Interarrival distribution
- Connection balance

Impact of the System Model



Impact of the Interarrival Distribution



Methodology

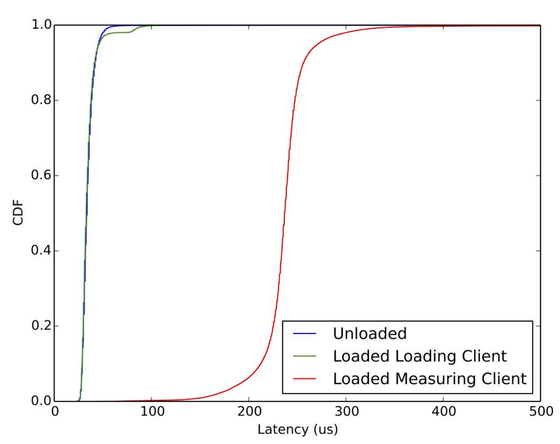
Feature	Potential Pitfall	Solution
Experiment Duration ● Warm up ● Measurement ● Cool Down	● Report unconverged results ● Run longer than needed	Online feedback loop that checks convergence and decides about the experiment duration
Measurement Collection: ● Symmetric ● Asymmetric (Separation of loading and measuring clients)	Uncaught behaviours: ● System imbalance ● Different latency across different request types ● Different behaviour across different connections	Symmetric models that exercise and account for every possible system behaviour
Statistical Aggregation: ● Aggregation over connection or client machine ● Histograms	● Coarse grained bucketing ● Load imbalance across connections	● Dynamic histograms ● Statistical aggregation over every connection
Heavy-tailed Distribution Identification	Underestimate latency because cases of tail latency are rare	Systematic heavy tail identification and warning about the accuracy of results

Tool

Requirements:
- Avoid client measuring bias
- Respect distributions

Solution:
- Identify interarrival distribution violations online
- Specify #clients using goodness of fit tests for the interarrival distribution

Impact of Loaded Clients



Tool Comparison

Tool/Feature	Distributed Coordination	Measuring Model	Dynamic Histograms
memaslap	No	Symmetric	No
YCSB	No	Symmetric	Yes
CloudSuite	No	Asymmetric	No
Mutilate	Yes	Asymmetric	No
TreadMill	No	Symmetric	Yes