

# Taming Performance Variability caused by Client-Side Hardware Configuration

Georgia Antoniou, Haris Volos, Yiannakis Sazeides  
University of Cyprus  
{gantoni12, hvolos01, yanos}@ucy.ac.cy

**Abstract**—Many online services running in datacenters are implemented using a microservice software architecture characterized by strict latency requirements. Consequently, this popular software paradigm is increasingly used for the performance evaluation of server systems. Due to the scale and complexity of datacenters, the evaluation of server optimization techniques is usually done on a smaller scale using a client-server model. Although the experimental details of the server side are excessively described in most publications, the client side is often ignored. This paper identifies the hardware configuration of the client side as an important source of performance variation that can affect the accuracy and the correctness of the conclusions of a study that analyzes the performance of microservices. This is partially attributed to the strict latency requirements of microservices and the small scale of the experimental environment.

In this work we present, using a widely used online-service, several examples where the accuracy and the trends of the conclusions differ based on the configuration of the client-side. At the same time we show that the experimental evaluation time can be significantly affected by the hardware configuration of the client. All these provoke the discussion of the right way to configure the experimental environment for assessing the performance of microservices.

**Index Terms**—performance variability, client-side, hardware configuration, microservices

## I. INTRODUCTION

Online applications running in today’s datacenters, such as social networks and web search, have moved from a monolithic to a microservice-based architecture. In this architecture, a monolithic application is decomposed into smaller, interconnected services that communicate explicitly with each other over the network through well-defined interfaces. These microservices can be independently developed, deployed, and scaled. However, due to the increased network overhead arising from the need for communication among services, each service must now adhere to stricter Quality-of-Service (QoS) constraints compared to its monolithic counterpart. Previous work reports tight QoS constraints for individual services, with 99th percentile latency targets that range from 250us to 500us [6], [7], [22], [49].

Given the rising prevalence of latency-critical applications based on microservices in today’s datacenters, the scientific community has increasingly turned to microservices to evaluate the performance of proposals targeting modern datacenter systems. This includes widely-used services, such as Memcached [1], which is typically deployed as a distributed caching service to accelerate user-facing applications [5], and

TABLE I: Hardware characterization in previous work.

| Characterization  | Publications |
|-------------------|--------------|
| Client only       | 0            |
| Server only       | 8            |
| Client and server | 2            |
| None              | 10           |
| Total             | 20           |

new benchmark suites, such as MicroSuite [38], DeathStar [14], and TrainTicket [52], which implement representative applications based on microservices.

Experimental evaluation utilizing the above frameworks typically entails deploying them on a small test cluster, following a client-server model. The test cluster often has fewer machine nodes compared to larger-scale production clusters, primarily due to complexity, scale, and cost constraints [4], [8], [26], [46]. Under this deployment, the test cluster comprises a set of server-side and client-side machines. Server-side machines are usually configured to host a few services rather than the whole application to keep the test cluster size under control while still achieving a representative setup. Client-side machines host the workload generator, which (i) generates a representative workload for the application to process, and (ii) accurately measures the end-to-end performance of the system under a target load, such as average response latency and tail latency (e.g., 99th percentile).

We observe that while experimental evaluations typically specify the server-side hardware configuration to ensure reproducibility of results, they often overlook the client-side configuration. Table I surveys the client- and server-side hardware configuration in recent publications (from the years 2021, 2022, and 2023) across various system and architecture conferences, including ISPASS, IISWC and MICRO. We find that only 10% of the papers studied specify the client-side hardware configuration. We attribute this limitation partially to the implicit assumption that the end-to-end response latency is dominated mainly by the server-side execution time. This assumption is rooted in past practices in experimental evaluations that were based on monolithic applications with millisecond-scale response latencies. However, this assumption no longer holds with microservices having microsecond-scale response latencies where any client-side microsecond-scale overhead can significantly impact the response latency. For example, waking up of a client-side processor core from a

power sleep state takes from 2us to 200us [48] (depending on the sleep state). This overhead can significantly impact the response latency of a microsecond-scale microservice, such as Memcached with an average server-side processing time of 10us [4], [7], resulting to an end-to-end response latency that can reach up to hundreds of microseconds.

To analyze the effect of client-side hardware configuration on performance evaluation, we conduct an experimental study based on representative, microservice-based services and applications with microseconds/few milliseconds response latencies, including Memcached [1], HDSearch from MicroSuite [38], Social Network from DeathStar [14] and synthetic workloads. Our experimental analysis reveals that client-side microsecond-scale hardware overheads, such as waking up from a power sleep state and dynamic voltage frequency scaling (DVFS [15]), can impact the accuracy of the end-to-end measurements leading to incorrect conclusions and additionally introducing performance variation. We find that the extent of the impact depends on a combination of (i) workload generator design, (ii) hardware configuration parameters, and (iii) server-side processing latency.

This behavior has several ramifications. In an academic setup, an analysis without consideration of the client-side hardware configuration can lead to inaccurate or wrong conclusions. At the same time, it renders the work unrepeatable as important details are missing from the experimental methodology of the paper. Finally, it degrades the validity of comparisons among techniques optimizing similar metrics in similar environments. In an industrial environment, performance evaluation is crucial for determining the load a machine can sustain without any QoS violations and guiding resource allocation for data centers [32], [33]. Ignoring client-side hardware configuration in this context can result in overprovisioning or underprovisioning of resources.

In summary, we make the following key contributions:

- We identify client-side configuration as a key source of performance variation in experimental evaluation.
- We demonstrate experimentally how and when client-side configuration can influence the accuracy and validity of the conclusions.
- We analyse the impact of different client-side configurations on the experimental evaluation time.
- We provide recommendations for how an experimental environment should be configured to improve representativeness and thus mitigate measurement inaccuracy caused by client-side configuration.

## II. CLIENT-CAUSED PERFORMANCE VARIABILITY

Due to the large scale of datacenters and web-based applications, researchers and practitioners typically evaluate data center related optimizations on test clusters with few nodes before propagating the optimization to the rest of the infrastructure. Measuring the performance of a service typically involves using a workload generator running on a set of client machines, as illustrated in Figure 1.

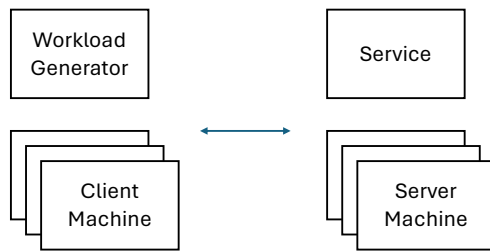


Fig. 1: Typical experimental methodology.

A workload generator is a software component that acts as a client that (i) generates requests for the service under study following a representative workload, and (ii) accurately measures the end-to-end latency (i.e., average latency, 99th percentile latency). Workload generators include the load intensity which represents the inter-arrival time of requests and resource demands which represents characteristics like the type and size of a request. Most previous work on experimental evaluation focuses on the workload generator (design and configuration), often neglecting the configuration of the client machines on which the generator runs.

Our key hypothesis is that client machine configuration can significantly impact workload and measurement accuracy, and the derived conclusions. The extent of the impact depends on a combination of (i) workload generator design, (ii) hardware configuration parameters, and (iii) service latency. Below, we qualitatively discuss how each such dimension may impact measurement inaccuracy. Later, in Section V, we present empirical evidence supporting our key premise.

**Workload generator design:** A workload generator timestamps generated requests and corresponding replies to model a target workload and measure end-to-end latency. This timestamp-based design makes the generator sensitive to timing inaccuracy in two ways. First, an open-loop generator models an infinite number of requests [24], sending requests to the target service according to an inter-arrival time distribution that represents the time between successive requests. Any inaccuracy in timing can disrupt the inter-arrival times, causing requests to shift in time and deviate from the target distribution. A closed-loop generator further limits the number of outstanding requests to model a finite number of blocking clients [24]. Because the timing of the next request depends on when the response to the previous request arrives, any timing inaccuracy can further impact the time when a successive request is sent. Overall, for both generator types, any timing inaccuracy can impact the timing of requests, causing the generated workload to deviate from the target workload.

Second, a workload generator can measure latency at various points in the system, such as the network interface card (NIC), the in-kernel socket layer, or the generator itself, collectively referred to as points of measurement [24].

With most typical workload generators, the measurement point resides within the workload generator itself. Therefore, measuring end-to-end latency depends on when the response reaches the generator and when the generator timestamps the response accordingly, rendering the measurement accuracy susceptible to any delay.

**Hardware configuration parameters:** The client-side hardware configuration refers to different configuration settings of the client-side system, including hyperthreading, turbo mode, C-states and CPU frequency. Such hardware settings can impact timing accuracy, potentially impacting both the generated workload and latency measurements in combination with the design of the workload generator.

For example, consider a time-sensitive workload generator with a point of measurement inside the generator itself, operating on a system with enabled c-states, allowing the system to sleep when idle. Upon issuing a request, the system may enter a sleep state until the corresponding response arrives. When the response arrives, the system must first wake-up and ramp up its frequency before the workload generator can timestamp the response and measure the end-to-end latency, consequently increasing the measured response time. Although this scenario may seem straightforward to avoid by configuring the client system to disable hardware features affecting timing accuracy to eliminate any variability, this approach may not always align with the target environment. For a target environment enabling c-states for low power, the point of measurement shall include any latency introduced by sleep state transitions. Otherwise, the experimental analysis may not be representative in terms of end-to-end latency. Since these types of analysis estimate the speedup of an optimization and ultimately guide the number of resources required to serve a target load, an inaccurate experimental environment may cause either overprovision or underprovision of resources. Unfortunately, enabling c-states with a time-sensitive workload generator to capture a representative point of measurement, may disrupt the generated workload, thus leading to conflicting choices.

Additionally to HW configuration parameters, kernel parameters (e.g., choice of idle governor), or compiler optimization flags can possibly cause similar effects on the accuracy of the end-to-end measurements. In this work we focus only on HW configuration parameters.

**Service latency:** With the emergence of the microservice software paradigm based on which applications are decomposed to several interconnected smaller services, the QoS of latency critical applications like search and social network has reduced significantly from milliseconds to microseconds while the request rates requirements have remained the same. Hardware overheads that are negligible for monolithic applications are now detrimental for the performance of microservice-based applications. As a result, microservices are especially vulnerable to the overhead introduced by the configuration of the client side since it is in the same order as the response time of the microservice (i.e., 250us). For example C-state transition overhead can take from 2us up to 200us depending on the

processor, while legacy DVFS takes several microseconds (i.e., 30us [15]). In the case of monolithic applications or microservice-based applications with higher response time the client side HW configuration shouldn't affect significantly the accuracy.

### III. STATISTICS PRIMER

In this section we present the background of the statistical methods used in our analysis.

**Confidence Intervals (CI):** When we display values for summarized datasets such as mean and average it is important to quantify their accuracy. In other words, since we gather empirical statistics in our experiments, confidence intervals (CI) [24], [25] offer some confidence that the empirical distribution collected experimentally is close to the actual distribution of the measured population. CI are ranges in which, we are x% sure that the population mean lies, where x represents the confidence level. A sampled mean of 20, x=95% and CI of 19.8 - 20.2, means that the true mean of the population distribution lies within 1% error from the estimated sampled mean. In order to be confident that a mean is higher than another, their CI should not overlap.

Depending on the distribution of the collected samples, we can either use a parametric or a non-parametric CI expression. Parametric expressions assume that the sampled data are derived from a known distribution (i.e., normal/Gaussian) whereas non-parametric expressions assume that the distribution of the sampled data is unknown. Many studies have demonstrated that data collected experimentally in computer systems, do not follow a normal distribution [21], [29], [47]. This is partially inline with what we have observed in our analysis (see Section V-C). To avoid assumptions of normality we use non-parametric confidence intervals (and other statistical methods) unless noted for the rest of the paper.

Non-parameric CI are computed based on the median instead of the mean. The following equations are used to compute the confidence intervals bounds for the median.

$$Lower\_bound = \lfloor \frac{n - z\sqrt{n}}{2} \rfloor \quad (1)$$

$$Upper\_bound = \lceil 1 + \frac{n + z\sqrt{n}}{2} \rceil \quad (2)$$

Where n is the number of samples in the set and z is the standard score which depends on the target confidence level. For a confidence level of 95%, z equals 1.96. Deriving confidence intervals involves first sorting the set of measurements, then using the above equations to determine the indices of the measurements corresponding to the lower and upper bounds of the confidence interval. The sample's median should be within the CI bounds.

**IID samples:** CI require the samples of a set to be independent and identically distributed (iid) [24], [25]. In the case of an experiment where the collection metric is latency, the samples are identically distributed since latency measurements come

from the same server. Regarding independence, in the analysis presented below we collect one sample per run. In between runs we reset the environment and as a result the measured samples are independent. When there is doubt for the iid-ness of samples, several methods can be used with the standard one being autocorrelation. Autocorrelation is a method that calculates the degree of similarity between a time series and a lagged version of itself. The output of the analysis can be anything between -1 and 1, where 1 represents a positive correlation, -1 a negative correlation and values close to 0 indicate no correlation among samples. Other methods used for assessing the iid-ness of samples include Lag-Plots and Turning Point Test.

**Hypothesis Testing - Shapiro-Wilk Test:** Hypothesis testing [27] is a systematic procedure used in statistics to assess whether characteristics of a population occur by chance or not. The first step in hypothesis testing, is to define a null hypothesis like for example two populations are equal. Then identify a test statistic that can evaluate the hypothesis. In our analysis we use a Shapiro-Wilk Test [37] in order to test whether the sampled data follow a normal distribution. Based on the test statistic results a p-value is calculated. P-values represent the probability of finding the observed results of a test statistic if the null hypothesis is true. The p-value is then compared with a significance level, if it is less than the significance level then we reject the null hypothesis. Conventionally 5% and 1% confidence levels have been used, which means that there is less than 1 in 20 and 1 in 100 chance of being wrong respectively.

**Sample Size for Determining Mean/Median:** The confidence level and accuracy of a CI depends on the number of samples. The higher the number of samples the better the associated confidence level and accuracy. In this section we describe 2 methods (1 parametric [18], 1 non parametric [29]) that can be used to determine what is the minimum required number of samples (repetitions in our case) that are required to achieve a confidence level with a certain accuracy.

Equation 3 [18] calculates the iterations for parametric distributions:

$$n = \left(\frac{100zs}{rx}\right)^2 \quad (3)$$

where  $z$  is the normal variate of the desired confidence level (1.96 for 95% confidence),  $s$  is the standard deviation,  $r$  is the error % from the mean and  $x$  is the mean of the collected samples.

For non-parametric distributions, the CONFIRM method [29] is used. To calculate the number of repetitions with CONFIRM: (i) for a set size  $n$ , randomly select a subset  $s \leq n$  and estimate non-parametric CI, (ii) shuffle set, select another subset, and estimate CI, (iii) repeat this procedure  $c$  times and then calculate the means for all the lower bounds of CI and upper bounds of CI, and (iv) if error is less or equal to 1% then size of the subset equals the number of repetitions, otherwise increase subset size and repeat. The original CONFIRM paper uses  $c=200$  and  $s \geq 10$  assuming

that smaller subsets cannot estimate non-parametric CIs reliably.

## IV. EXPERIMENTAL METHODOLOGY

### A. System

To conduct our experiments we use the c220g5 cluster of the Wisconsin site from the CloudLab [13] infrastructure. Our baseline system is a 2 socket server with 2 Skylake-based (Intel Xeon Silver 4114) processors. There are 20 physical cores and 40 hardware threads. The nominal frequency is 2.2GHz with the minimum frequency reaching 0.8 GHz and the maximum Turbo Boost frequency 3 GHz. The server is equipped with 192 GB DDR4 DRAM. The operating system used is UBUNTU 18.04.

### B. Benchmarks

Two representative microservice-based latency critical applications are used in the analysis:

**Memcached** [1] is a lightweight key-value store that is widely deployed as a distributed caching service to accelerate user-facing applications with strict latency requirements. Memcached has been the focus of numerous studies, including efforts to provide low microsecond-scale tail latency. In our experiments, we run a memcached instance with 10 worker threads pinned on a single socket. We use an extended version of Mutilate [26], as a workload generator. Following the taxonomy of Section II, Mutilate is an open-loop workload generator; it implements time-sensitive interarrival times using a block-wait event loop that waits for elapsed time, with the point of measurement residing within the generator itself. We run Mutilate on 5 machines, one for the master process and 4 for the workload-generator clients, establishing a total of 160 connections. We configure the workload generator to recreate the ETC workload from Facebook [5].

**HDSearch** [38] is one of the four information-retrieval services of the MicroSuite microservice-based benchmark suite. HDSearch is an image similarity search service written in C++, which is structured as a three-tier service using RPC for communication between tiers. It returns images from a large dataset whose feature vectors are near to the query's feature vector. It uses Locality-Sensitive Hash (LSH) tables to traverse the search space of the problem efficiently. We use the accompanying open-loop client, which generates requests with inter-arrival times drawn from a Poisson distribution, as a workload generator. Following the taxonomy of Section II, the client is an open-loop workload generator; it implements time-insensitive interarrival times using a busy-wait loop that actively polls for elapsed time, with the point of measurement residing within the generator itself. We use 3 machines to run the benchmark, 1 for each type of process: client, midtier and bucket. Our benchmark configuration follows the configuration of the MicroSuite paper [38]. Finally, we pin the processes onto specific cores to eliminate process migration. **Social Network** is a microservice-based application from the DeathStar [14] benchmark suite, consisting of multiple interconnected services. We deploy the benchmark on a single

node using Docker Swarm. We initialize the social graph using the provided small dataset namely "Reed98 Facebook Networks" [36]. We use the accompanying open-loop client, which is an extended version of the wrk2 workload generator. We configure the client to (i) establish 20 connections with the server, (ii) send requests using an exponential distribution, and (iii) only use read-user-timeline requests. Following the taxonomy of Section II, the client is an open-loop workload generator; it implements time-sensitive interarrival times using a block-wait event loop that waits for elapsed time, with the point of measurement residing within the generator itself. Finally, before each run we fill the database of the application with posts using compose-post queries.

**Synthetic Workload** is a program with tunable service latency, implemented to perform a sensitivity analysis. It can accept an input parameter, the value of which specifies by how long the processing time of a request should be extended. The processing time is implemented using a busy wait loop instead of a normal wait loop to prevent the core from serving other requests, as the additional wait time should be accounted as service time rather than sleep time. We run our service instance with 10 worker threads pinned on a single socket. Following the taxonomy of Section II, the client of the synthetic workload, is an open-loop workload generator; it implements time-sensitive interarrival times using a block-wait event loop that waits for elapsed time, with the point of measurement residing within the generator itself.

Unless stated otherwise, each experiment is the average of 50 runs. The duration of each run is 2 minutes. We collect several metrics during the execution of each experiment with the most important one being the average response time and 99th tail latency. We use the non-parametric expressions to calculate CI with a confidence level of 95%. In each experiment we tune several hardware knobs. The description of each HW knob is mentioned in Section IV-C and the scenarios evaluated are mentioned in Section IV-D.

### C. Hardware Knobs

In this section we describe the different HW knobs of the analysis and how we tune them.

**C-states** [16] are power saving states that enable a core to reduce its power consumption during idle periods. Skylake-based processors support 4 C-states C0, C1, C1E and C6. We use the intel\_idle.max\_cstate flag and the idle=poll flag to enable/disable any C-states through the grub file.

**Frequency Driver** [23] is a component of the CPUFreq subsystem of Linux that enables the OS to scale the frequency and voltage. The frequency driver is responsible for communicating the Frequency/Voltage settings to the hardware. Usually a linux system supports 2 frequency drivers, intel\_pstate and acpi-cpufreq. We pass the intel\_pstate flag to the grub file to enable/disable them.

**Frequency Governor** [23] is also a component of the CPUFreq subsystem. It is the component responsible to decide the suitable frequency/voltage of the system based on some heuristics. We use cpupower, which is a tool that act as

a wrapper around the sysfs kernel interface to specify a frequency driver.

**Turbo mode** [16] is a feature in modern processors that allows CPU to dynamically increase its clock speed above its nominal frequency under certain conditions (i.e., thermal capacity, number of active cores). We use the Model Specific Register (MSR) 0x1a0 to enable/disable turbo mode.

**Simultaneous Multithreading (SMT)** [42] is a feature in modern processors that allow multiple threads to execute on the same physical core at the same time. We use the sys interface to enable/disable this feature.

**Uncore Frequency** [16] refers to the operating frequency of the uncore components of the CPU. These components include the Last Level Cache (LLC), IO interfaces etc. We use the MSR 0x620 to tune the uncore frequency.

**Tickless** [40] is a characteristic of kernels that do not omit clock-scheduling interrupts during idle periods. We pass the nohz flag to the grub file to enable/disable this feature.

### D. Client/Server Configuration

In our analysis we use 2 configurations for the client-side, the low-power (LP) configuration and the high-performance (HP) configuration. The LP configuration represents the default configuration of the system and thus the case where a user is agnostic of the client-side configuration. The HP configuration represents a configuration tuned empirically to achieve high performance. The details of the configuration can be found in Table II.

The server side baseline configuration is presented also in Table II. We choose empirically a configuration that does not introduce high variability and achieves good performance. In the experimental evaluation whenever a HW knob of the server side changes it is explicitly mentioned.

TABLE II: Client- and server-side hardware configurations

| Configuration             | Client Side  |              | Server Side  |
|---------------------------|--------------|--------------|--------------|
|                           | LP           | HP           | Baseline     |
| <b>C-states</b>           | C0,C1,C1E,C6 | off          | C0,C1        |
| <b>Frequency Driver</b>   | intel pstate | acpi cpufreq | acpi cpufreq |
| <b>Frequency Governor</b> | powersave    | performance  | performance  |
| <b>Turbo</b>              | on           | on           | off          |
| <b>SMT</b>                | on           | on           | off          |
| <b>Uncore Frequency</b>   | dynamic      | fixed        | fixed        |
| Tickless                  | off          | off          | on           |

Table III describes the scenarios tested in the experimental analysis (Section V) using the terminology introduced in Section II. The last column of the table, indicates which one of the scenarios can cause wrong conclusions (e.g. X) and the sections each scenario is evaluated in.

## V. EXPERIMENTAL ANALYSIS

We study the impact of client-side hardware configuration on performance variation under different scenarios (Section V-A) and how this impact varies with services with

TABLE III: Scenarios Tested in Section V.

| Workload Generator Design  |                | Client Conf. | Response Time | Risk/Section |
|----------------------------|----------------|--------------|---------------|--------------|
| inter. rate                | point of meas. |              |               |              |
| open-loop time-sensitive   | in-app         | tuned        | small         | (5.1,5.3)    |
| open-loop time-sensitive   | in-app         | not-tuned    | small         | X(5.1,5.3)   |
| open-loop time-insensitive | in-app         | tuned        | big           | (5.2)        |
| open-loop time-insensitive | in-app         | not-tuned    | big           | (5.2)        |

higher response time (Section V-B). Finally, we examine the impact of the client-side configuration on the execution time of the experimental evaluation (Section V-C).

#### A. Client-side Configuration Impact

We present two case studies that aim to evaluate the impact of two server-side features, specifically SMT and C-states, on the performance of the Memcached service. Our findings demonstrate that the choice of client-side configuration can lead to varying performance results and differing conclusions regarding the effects of the features under study.

**SMT:** The aim of the analysis is to investigate whether SMT can improve the performance of Memcached under different load (10K - 500K QPS) and corresponding utilization (5% - 55%). Figure 2 shows the performance evaluation of Memcached running on a server machine configured with SMT disabled (baseline) or SMT enabled. Each server-side hardware configuration is examined with two client configurations, namely LP (low power) and HP (high performance).

Depending on the client configuration, the end-to-end measurements differ. Specifically, the LP end-to-end measurements are between 80% to 150% higher than the end-to-end measurements of HP client, that is if we compare similar server-side configurations. Additionally, the 99th percentile latency is 33% to 200% higher for LP clients compared to HP clients. We argue that this is a result of the additional overhead introduced by the client-side hardware configuration. Since the point of measurement of the workload generator is inside the generator itself, a query must experience at least a C-state transition (2us - 200us), a DVFS transition ( $\sim 30$ us), and a context switch ( $\sim 25$ us) before the workload generator is able to capture the timestamp that will mark the completion of the query. This behavior is especially important in a datacenter setup. Let us assume a service with a QoS of 99th percentile latency equal to 400us. The LP client finds that the service can handle only 300K queries without violating any QoS constraints. In contrast, the HP client finds that the service can handle 500K queries. In other words, the LP client determines that a deployment will require 1.6x more machines than the HP client, to satisfy the same load without violating any QoS constraints.

Another important observation from Figure 2c and Figure 2d is that the measured degradation depends on the client-side hardware configuration. The LP client determines

that enabling SMT on the server side improves the 99th percentile latency by at most 3% (see Figure 2d). In contrast, the HP client determines that the 99th percentile latency can improve by 13%. We believe this is partially because the absolute performance improvement caused by SMT is more pronounced for the HP client end-to-end time compared to the LP client.

**Finding 1:** The client-side hardware configuration can impact the accuracy of an experiment. Specifically, it can (i) affect the end-to-end measurements, leading to higher or lower measurements, and (ii) produce different speedups for the same feature or technique under study.

**C1E:** The aim of the analysis is to investigate whether C1E can improve the performance of Memcached under different load (10K - 500K QPS) and corresponding utilization (5% - 55%). Figure 3 shows the performance evaluation of Memcached running on a server machine configured with C1E disabled (baseline) or C1E enabled. Each server-side hardware configuration is examined with two client configurations, namely LP (low power) and HP (high performance).

Similarly to the SMT study above, the choice of client configuration leads to different end-to-end average response latency and 99th percentile latency. Specifically, the average response latency differs from 64% to 145% and the 99th percentile latency from 0% to 200%. Additionally, the observed slowdown caused by C1E differs based on the client configuration. For the HP client, the slowdown of C1E goes up to 19% for average latency and 18% for the 99th percentile latency. For the LP client, the slowdown caused by C1E goes up to 13% for the average latency and 7% for the 99th percentile latency.

More importantly, the client choice shows different trends for high load (400K and 500K QPS), leading to conflicting conclusions about the effect of C1E on performance. The LP client reports that for high load the C1E enabled configuration is worse than the C1E disabled (since the confidence intervals do not overlap). However, the HP client reports that for all loads (except of the 10K QPS load) the C1E enabled and C1E disabled configurations have the same performance.

**Finding 2:** The client-side hardware configuration can impact not only the accuracy but also the observed trends of an experiment, leading to conflicting conclusions.

#### B. Impact Relative to Service Latency

We examine the impact of client-side hardware configuration on the performance of applications with different end-to-end latencies. We present three studies: (i) a single-service study, which investigates the performance of a microservice-based service benchmark, (ii) a multi-service application study, which investigates the performance of a microservice-based application consisting of multiple services, and (iii) a synthetic workload study, which performs a sensitivity analysis. Our findings demonstrate that the client-side hardware configuration has minimal impact on services with high response latency.

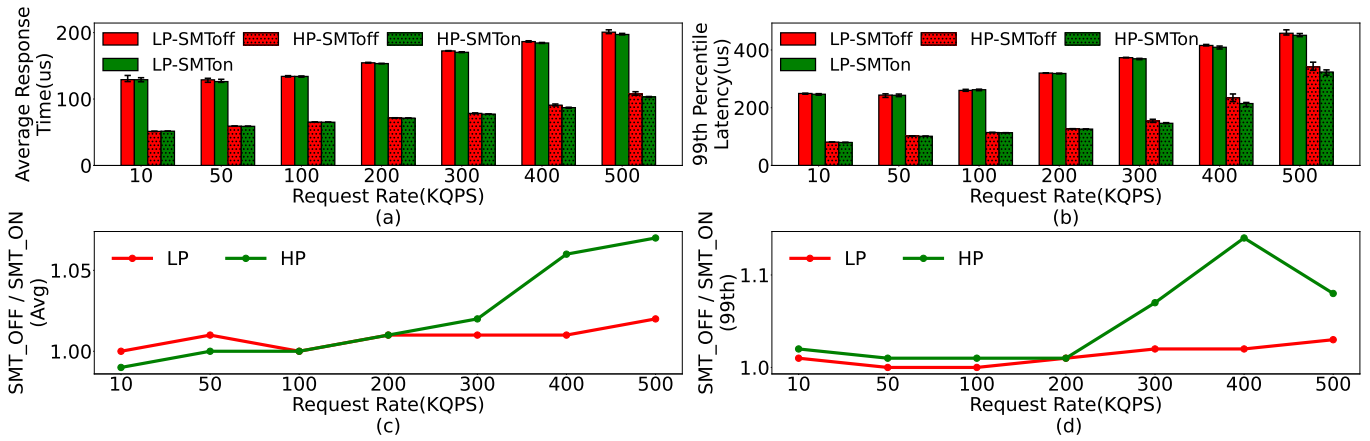


Fig. 2: Performance evaluation of SMT impact on Memcached service latency with LP and HP clients. (a) Average Response Time (median) for HP/LP client and SMT ON/OFF server, (b) 99th Percentile Latency (median) for HP/LP client and SMT ON/OFF server, (c) Slowdown (avg) caused by disabling SMT on the Average Response Time for HP and LP client and (d) Slowdown (avg) caused by disabling SMT on the 99th Percentile Latency for HP and LP client.

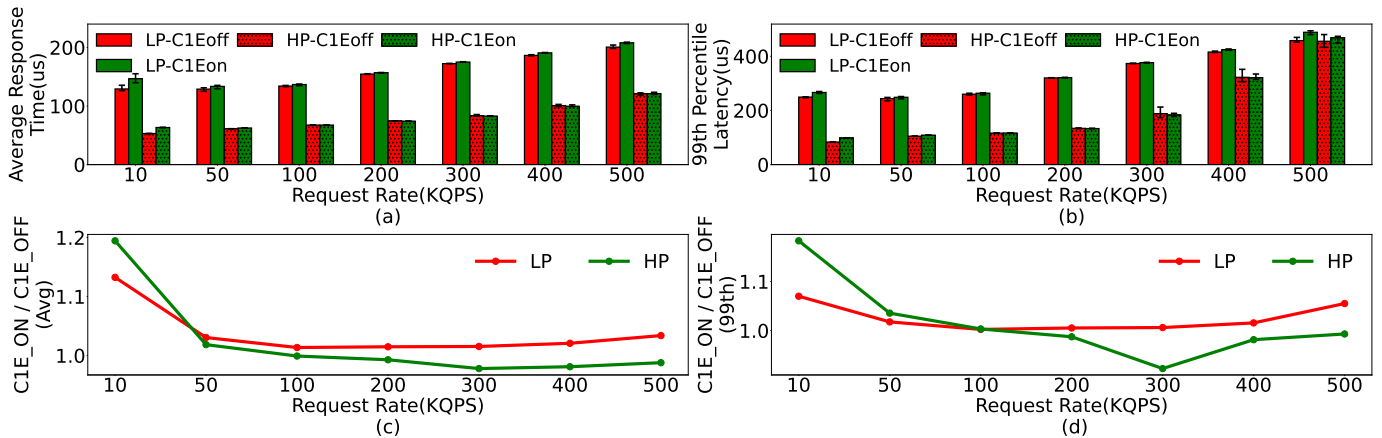


Fig. 3: Performance evaluation of C1E impact on Memcached service latency with LP and HP clients. (a) Average Response Time (median) for HP/LP client and C1E ON/OFF server, (b) 99th Percentile Latency (median) for HP/LP client and C1E ON/OFF server, (c) Slowdown (avg) caused by enabling C1E on the Average Response Time for HP and LP client and (d) Slowdown (avg) caused by enabling C1E on the 99th Percentile Latency for HP and LP client.

**Single-Service:** We use the HDSearch service, which operates with millisecond-scale latency, to examine the impact of client-side configuration on performance variation when analyzing the performance of services with high response latency. Figure 4, presents the performance evaluation of HDSearch running on a server machine configured with SMT or C1E. Each server-side HW configuration is examined with two client configurations, namely LP (low power) and HP (high performance).

Similarly to Memcached, there is a difference in end-to-end measurements between the HP and LP client for both average and 99th percentile latency, although it is not as pronounced as in Memcached. Specifically, the average response latency of LP is from 7% to 17% higher than HP. Regarding the 99th percentile latency, LP has 5% to 29% higher 99th percentile

latency than HP. Since HDSearch has higher response latency than Memcached, we expect the difference between the end to end measurements of the HP and LP clients which is a result of the client configuration overhead, to be statistically less significant. Thus, we expect LP and HP clients determine similar resource requirements to satisfy a target load without violating any QoS constraints.

Contrary to Memcached, the HP and LP clients measure same speedups (with similar trends) in the average response latency for both the SMT and C1E server-side configurations. Even though the LP measurements experience variability because of the client-side hardware configuration, the high server-side processing time of HDSearch (400us) overshadows the client-caused variability ( $\sim 20\text{us}$  in Figure 5b), thus minimally impacting the observed speedup of the evaluated

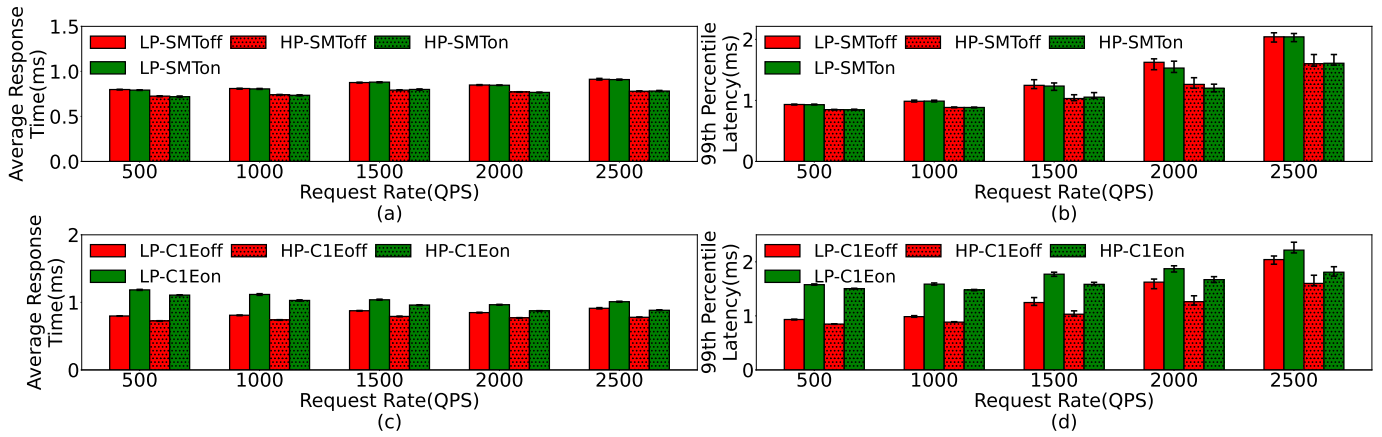


Fig. 4: Performance evaluation of SMT and C1E impact on HDSearch service latency with LP and HP clients. (a) Average Response Time (median) for HP/LP client and SMT ON/OFF server, (b) 99th Percentile Latency (median) for HP/LP client and SMT ON/OFF server, (c) Average Response Time (median) for HP/LP client and C1E ON/OFF server and (d) 99th Percentile Latency (median) for HP/LP client and C1E ON/OFF server.

server configurations. Memcached server-side processing time ( $\sim 10\mu s$ ) is in the same order as the client-caused variability (up to  $10\mu s$  in Figure 5a), thus making the speedup of the evaluated server configurations more sensitive to the client-side hardware configuration.

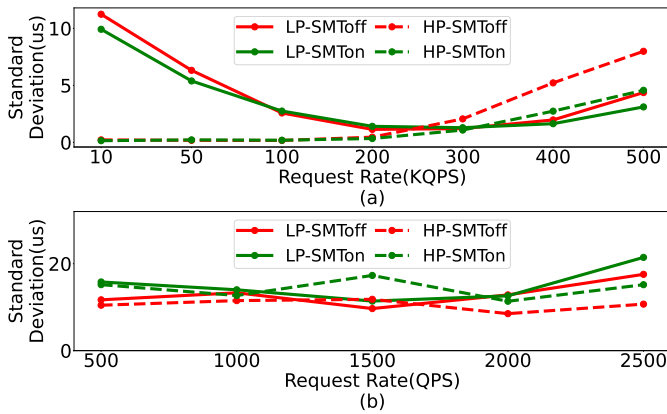


Fig. 5: (a) Standard Deviation of Memcached for the Average Response Time with LP/HP client configuration and SMT ON/OFF server configuration, (b) Standard Deviation of HDSearch for the Average Response Time with LP/HP client configuration and SMT ON/OFF server configuration.

Overall, we observe that HDSearch, a service with about 10 times higher end-to-end response latency than Memcached exhibits similar speedups and trends when run with two different client configurations. Although the absolute end-to-end measurements are not the same, the difference is not as pronounced as in Memcached.

**Multi-Service Application:** We study the impact of the client-side HW configuration on the performance of an application using Social Network from the DeathStar benchmark

suite. Figure 6a presents the difference in the end-to-end latency between the two client configurations LP and HP for average and 99th percentile latency respectively. Similarly to HDSearch the difference between the two clients gets smaller while the end-to-end latency increases. Compared to HDSearch, the gap between the two clients is small (5% vs 17%) on the average response time due to the fact that Social Network has higher end-to-end latency ( $\sim 2-3ms$  in Figure 6b) than HDSearch ( $\sim 1ms$  in Figure 4). Surprisingly the impact of different clients on the 99th percentile latency for Social Network, as shown in Figure 6c, is minimal. In other words, the 99th percentile reported by both clients, LP and HP is the same. For end-to-end latencies over  $10ms$ , the client-induced overhead does not appear to affect the accuracy of the measurements, as illustrated in Figure 6c.

Overall, we observe that DeathStar validates the HDSearch analysis. Although the absolute measurements reported by the two clients are not identical, the difference is not as pronounced as in Memcached and HDSearch.

**Synthetic Workload:** To examine the impact of client side HW configuration at different latencies, we use the synthetic workload. Figure 7a and Figure 7b present the performance evaluation of the synthetic workload for different end-to-end latencies and QPS under two client configurations LP and HP. Although the QPS presented in Figure 7 are low compared to the ones examined in previous sections, it is important to note that due to the increase in processing time there is no opportunity to achieve higher throughput. To determine the examined QPS we use Little's law and examine only the QPS where the concurrency is less than the number of available cores (i.e., 10) for all possible values of the new parameter. Additionally, the results presented in this section are the average of 20 runs.

As expected with the increase of the end-to-end latency, the gap between the LP and HP reported end-to-end mea-



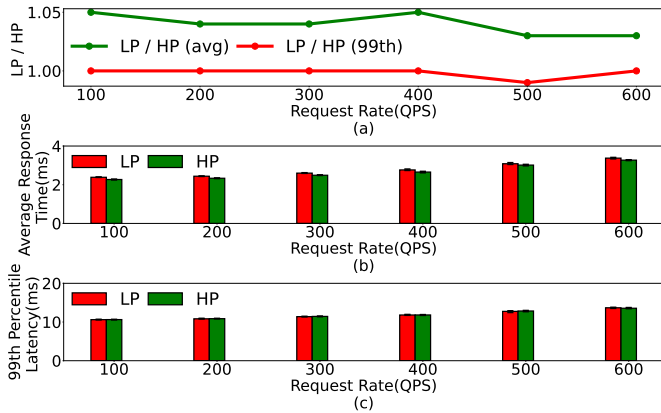


Fig. 6: Performance evaluation of HP and LP clients for Social Network. (a) Slowdown (avg) caused by changing from HP to LP client on the Average Response Time and 99th Percentile Latency, (b) Average Response Time Latency (median) for HP/LP client at different QPS and (c) 99th Percentile Latency (median) for HP/LP client at different QPS.

measurements gets smaller. Specifically, the difference goes from 2.8x for 0 added delay to 1.02x for 400 us added delay at 20K QPS. Similarly, for 99th percentile latency the difference goes from 3.5x to 1x. Between 0 to 100us added delay we observe the highest decrease in the difference between the end to end measurements reported by HP and LP client for both average and 99th percentile. This analysis confirms the findings of HDSearch, based on which the added client side overhead becomes less statistically significant for benchmarks with higher latencies.

Figure 7c, 7d, 7e and 7f demonstrate the absolute end-to-end measurements for HP and LP client at 5K and 20K QPS. At low QPS, where there is no queueing, the response time increases linearly with the increase of the added delay which validates the implementation of the synthetic workload. We observe that for average response time latencies over 1ms the accuracy difference is less than 10% between HP and LP client. For high QPS and high added delays (end to end over 2ms) the HP and LP clients measurements converge. This is partially because of the variability of the experiment being in the same order as the observed overhead introduced by the client side hardware configuration ( $\text{stdev} \sim 100\text{us}$ ). A major source of variability for high QPS is the queueing caused in the server side. We conclude that when the end to end latency is in the order of milliseconds, the impact of client side overhead is less significant.

**Finding 3:** The client-side hardware configuration has minimal impact on services with high response latency. The client-side hardware configuration causes performance variability when the processing time of an application is in the same order of magnitude as the variability introduced by the client side.

### C. Impact on Experimental Evaluation Time

In this section, we investigate how the different client-side hardware configurations affect the experimental evaluation time. By experimental evaluation time, we mean the time required for an experiment to achieve a confidence interval with at most 1% error at a 95% confidence level. Before estimating the number of repetitions an experiment requires to gain statistical confidence, we first check whether the collected samples follow a normal distribution. This is because the closed-form expressions used to calculate the number of iterations for an experiment assume that data follow a normal distribution.

Figure 8 tests the normality of the data presented earlier in Section V-A using the Shapiro-Wilk test. Data points within a single configuration correspond to varying loads (QPS), all collected from the same single server. The red dashed line indicates the threshold below which configurations do not conform to a normal distribution. We analyze a total of 42 configurations (six scenarios each with seven QPS values), with each configuration comprising 50 runs. Approximately 50% of these configurations adhere to a normal distribution, while the remaining 50% do not.

The above normality test results are in line with previous work that examines data normality on a single node. Specifically, within the LP-SMToff scenario, all QPS configurations exhibit a normal distribution. Conversely, none of the QPS configurations within the HP-SMTon scenario adhere to a normal distribution. In the HP/LP-SMToff and HP/LP-C1Eon scenarios, approximately half of the QPS configurations follow a normal distribution, while the remaining half corresponding to the high QPS configurations do not adhere to a normal distribution. In the HP/LP-SMToff and HP/LP-C1Eon scenarios, about half of the QPS configurations conform to a normal distribution, whereas the other half, comprising the high QPS configurations, do not. We attribute this non-normality to queuing effects that are more pronounced for higher QPS and the reduced number of logical threads (SMToff). Looking into the frequency charts of these high QPS configurations, a large number of samples lies below and close to the median of the distribution, whereas a small number of samples is scattered in a larger range above the median, making the distribution skewed, as shown in Figure 9.

Based on the above normality test results, we use both parametric and non-parametric (CONFIRM) methods to calculate the number of required iterations to achieve a confidence interval with at-most 1% error and 95% confidence level for each configuration (as explained in Section III).

Table IV presents the results of the two methods along with the Shapiro-Wilk test results. The highest value of iterations estimated by CONFIRM is  $>50$  since each experiment is executed 50 times. The lowest value estimated by CONFIRM is 10, since the method assumes that smaller subsets cannot estimate non-parametric CIs reliably. The two methods do not produce exactly the same results partly due to the parametric method's ability to reliably estimate values and provide tight

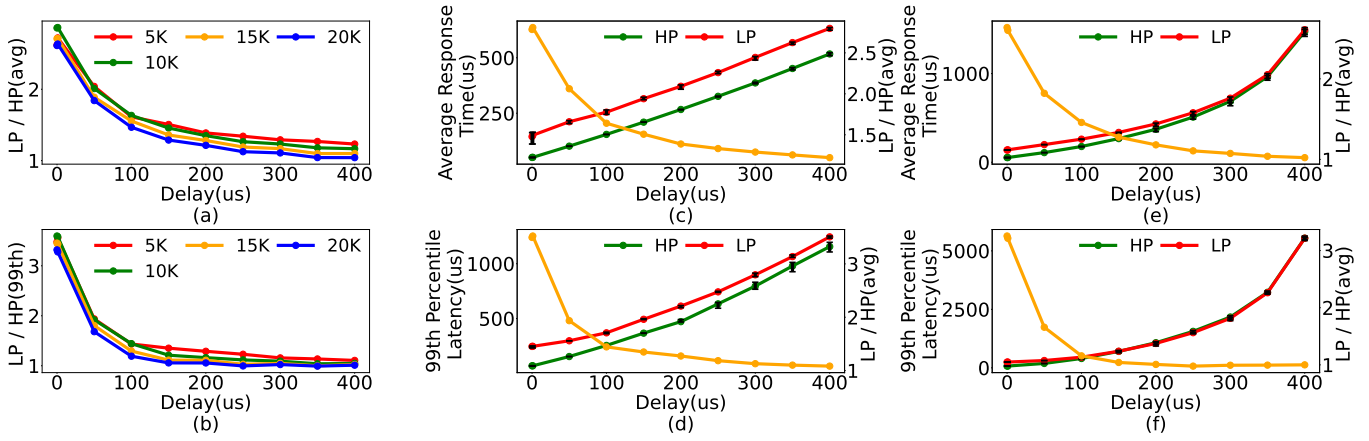


Fig. 7: Performance evaluation of HP and LP clients for different processing times. (a) Slowdown (avg) caused by changing from HP to LP client on the Average Response Time, (b) Slowdown (avg) caused by changing from HP to LP client on the 99th Percentile Latency, (c) Average Response Time (median) for HP/LP client at 5K QPS, (d) 99th Percentile Latency (median) for HP/LP client at 5K QPS, (e) Average Response Time (median) for HP/LP client at 20K QPS and (f) 99th Percentile Latency (median) for HP/LP client at 20K QPS.

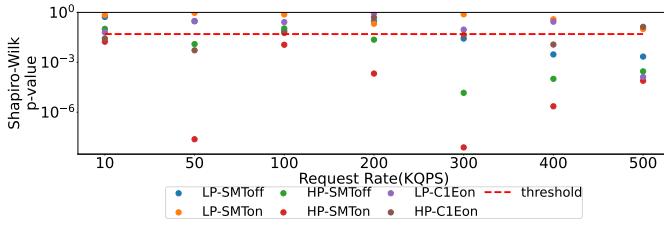


Fig. 8: Shapiro-Wilk p-value for configurations in Section V-A.

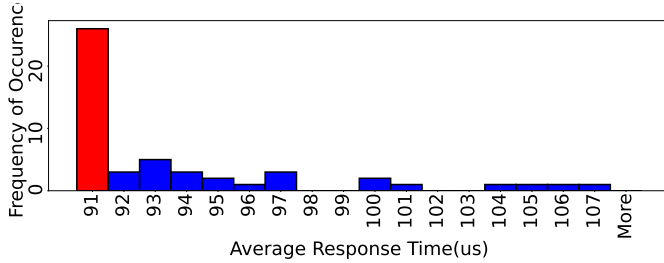


Fig. 9: Frequency Chart for HP-SMToff 400K configuration. The red bar is where the median lies.

bounds with a fewer number of iterations, typically below 10, when the configuration adheres to a normal distribution. As a result, there are several cases where the parametric method estimates just one iteration, and the CONFIRM method requires 10 iterations.

Nevertheless, the two methods support that different client configurations require different number of iterations to produce tight statistically confident results. For low QPS (10K - 100K), both methods agree that the LP client requires a large number of iterations to achieve statistical confidence

whereas the HP client requires much less. For high QPS (300K - 500K), the HP client requires more iterations than the LP client. This behavior agrees with our empirical observations, that want the LP client to have higher standard deviation in low QPS than the HP client, and the HP client to have higher standard deviation in high QPS (see Figure 5).

**Finding 4:** The client-side hardware configuration can affect the number of iterations needed for an experiment because different configurations can exhibit different levels of performance variability. Current experimental methods are effective at estimating the number of iterations required to mitigate the performance variability caused by the client.

## VI. CONFIGURATION RECOMMENDATIONS

Drawing from the taxonomy of Section II and the experimental analysis of Section V, we now discuss recommendations for how to best configure the client side in an experimental evaluation based on latency-sensitive microservices. We focus on the aspect of time-sensitivity caused by the interarrival time implementation of open-loop workload generators, as we find this can play a key role in performance variation.

For a time-sensitive interarrival time implementation, the client-side hardware configuration should be tuned for performance. The performance configuration mitigates the hardware timing overheads of power and energy optimizations (*i.e.*, C-states, DVFS), allowing the workload generator to send requests as close as possible to the time indicated by the interarrival time distribution. In this case however, it is essential to consider how accurately the performance configuration reflects the configuration within the target production cluster. If the configuration deviates from the target production configuration, then it may over- or under-estimate

TABLE IV: Number of iterations to gain statistical confidence and Shapiro-Wilk results.

| Configuration | QPS  | Parametric | CONFIRM | Shapiro-Wilk |
|---------------|------|------------|---------|--------------|
| LP-SMToff     | 10K  | 288        | >50     | pass         |
|               | 50K  | 93         | >50     | pass         |
|               | 100K | 15         | 37      | pass         |
|               | 200K | 3          | 11      | pass         |
|               | 300K | 2          | 11      | fail         |
|               | 400K | 5          | 19      | fail         |
|               | 500K | 19         | >50     | fail         |
| LP-SMTon      | 10K  | 225        | >50     | pass         |
|               | 50K  | 70         | >50     | pass         |
|               | 100K | 17         | 34      | pass         |
|               | 200K | 4          | 16      | pass         |
|               | 300K | 3          | 11      | pass         |
|               | 400K | 4          | 15      | pass         |
|               | 500K | 10         | 36      | pass         |
| HP-SMToff     | 10K  | 1          | 10      | pass         |
|               | 50K  | 1          | 10      | fail         |
|               | 100K | 1          | 10      | pass         |
|               | 200K | 2          | 11      | fail         |
|               | 300K | 27         | >50     | fail         |
|               | 400K | 123        | >50     | fail         |
|               | 500K | 203        | >50     | fail         |
|               | 500K | 203        | >50     | fail         |
| HP-SMTon      | 10K  | 1          | 10      | fail         |
|               | 50K  | 1          | 10      | fail         |
|               | 100K | 1          | 10      | fail         |
|               | 200K | 1          | 10      | fail         |
|               | 300K | 8          | 11      | fail         |
|               | 400K | 39         | 41      | fail         |
|               | 500K | 77         | 41      | fail         |
| LP-C1Eon      | 10K  | 303        | >50     | pass         |
|               | 50K  | 89         | >50     | pass         |
|               | 100K | 20         | >50     | pass         |
|               | 200K | 3          | 11      | pass         |
|               | 300K | 1          | 10      | pass         |
|               | 400K | 2          | 11      | pass         |
|               | 500K | 9          | 21      | fail         |
| HP-C1Eon      | 10K  | 2          | 11      | fail         |
|               | 50K  | 1          | 10      | fail         |
|               | 100K | 1          | 10      | pass         |
|               | 200K | 1          | 10      | pass         |
|               | 300K | 8          | 24      | fail         |
|               | 400K | 21         | >50     | fail         |
|               | 500K | 32         | >50     | pass         |

performance metrics, such as end-to-end time (Section V-A), and consequently affect any conclusions drawn, such as those related to resource provisioning.

For a time-insensitive interarrival time implementation, the choice is guided by the target environment. The configuration of the client should match the configuration of the target environment. When the target configuration is unknown, a space exploration could be made to evaluate a technique under several scenarios, using either homogeneous or heterogeneous client and server machine configurations.

As far as the number of iterations required for an experiment is concerned, well established methodologies [18], [29] should be used based on the distribution followed by the samples.

## VII. RELATED WORK

To the best of our knowledge, this is the first study to investigate the impact of the client-side hardware configuration on the accuracy and evaluation time of an experiment. Previous works focus on quantifying variability arising from

other sources, including the order of experiments, process variation, and the server-side configuration. Several studies propose techniques to mitigate variability, including increasing the number of repetitions, using confidence intervals, and developing more robust workload generators.

### A. Evaluating Performance using Microservices

In recent years, latency-critical applications have moved from a monolithic to a microservice-based software architecture to satisfy service-level objectives, availability, scalability, and regular updates [19], [35], [43]. In a microservice-based software architecture, an application is decomposed into several services that communicate with one another via the network through well-defined interfaces, such as gRPC and REST APIs. The decoupled nature of these applications leads to stricter QoS constraints per service compared to their monolithic counterparts, ranging from 250us [7], [49] to 500us [6], [22]), due to increased network communication overheads.

The transition to a microservice software paradigm has prompted the community to develop new benchmark suites, such as MicroSuite [38] and DeathStar [14], and adopt existing services, such as Memcached [1], to effectively evaluate designs targeting microservices. Memcached, in particular, has been the focus of numerous studies, including tail-latency optimizations [30], collocation [26], request scheduling and consolidation, and C-states [3], [4], [48], due to its critical role in enhancing response times of latency-critical applications as a lightweight caching service [28]. To simplify the experimental environment and facilitate reproducibility, deploying a single memcached server process for the experimental evaluation has been common practice among previous works [4], [8], [26], [46].

### B. Quantifying Performance Variability

Quantifying variability has been the focus of many works on datacenters, supercomputers and smartphones. Maricq *et al.* [29] investigate what is the inevitable variability across nodes of the same architecture in a cluster. They conclude that variability of up-to 10% can be attributed to the underlying hardware. Additionally, they investigate the normality of performance samples across nodes and conclude that the performance samples follow a non-parametric distribution across nodes. In a similar setup, Duplyakin *et al.* [12] investigate the variability caused by the execution order of experiments. The rationale is that the sequence in which experiments are conducted can alter the microarchitectural characteristics of the machine, inevitably affecting the performance outcomes of each experiment. If executed in a specific order, this bias will impact the outcome of the experiment. Such variability can be categorized as a form of measurement bias. A measurement bias [31] is when a technique X speedups a system O by Z but the speedup is not only a result of the technique but is also a bias of the experimental setup. Several works have investigated this phenomenon in various settings, including scheduling algorithms of supercomputers [41], architectural

simulations for multithreaded workloads [2], and O3 optimizations in SPEC CPU2006 workloads [31]. Additionally to measurement bias, other works [44], [47] have identified the network contention as a major contribution to the variability observed by an application. Another work [20] has developed a methodology using the stress-ng [9] tests that can estimate the variability across machines of different architecture and as a result can reproduce with some error an experiment outcome on a different machine. Finally, Srinivasa *et al.* [34] manage to create a methodology that quantifies the process variation of smartphones at system level. Although different sources of performance variability have been investigated there is no mention of the configuration of the client side, even in survey papers [12], [17], even though it can affect the accuracy of measurements.

### C. Mitigating Performance Variability

Strategies aiming to improve the experimental methodology accuracy and mitigate the performance variability have been proposed, many of which have been implemented inside a workload generator [10], [24], [26], [50]. For example in Lancet [24], the authors try to create a workload generator that accurately captures the 99th percentile latency of microservices by (i) minimizing the errors caused by excessive user interference, (ii) using state of the art hardware-based techniques, and (iii) using excessively statistical methods to accurately process the samples. Their workload implements, among others, an Anderson Darling test to check the request arrival distribution, an Augmented Dickey Fuller test to check the stationarity of samples, and a Spearman test to check whether samples are independent. Apart from workload generators, standalone tools have also been proposed, such as CONFIRM [29] and OrderSage [12], aiming to mitigate variability, in this case, by (i) calculating the CI for non-parametric distributions, and (ii) randomizing the order of experiments. Another set of works [11], [45], [51] fall under the umbrella of variability detection (anomaly detection or changepoint detection). Other works aim to mitigate measurement bias by randomizing the experimental setup, either through slight changes in the timing of requests of the workload [41] or modifications in the simulator to change cache-miss penalty [2]. Finally, request batching [39] has been proposed for eliminating network variability in Memcached. The proposals mentioned above are complementary to our work.

## VIII. CONCLUSIONS

To the best of our knowledge, this is the first work that examines the impact of the client-side configuration on the experimental evaluation of microservices. Our evaluation reveals that under certain conditions that concern the design of the workload generator and the characteristics of a microservice, the client-side configuration can influence the accuracy of the end-to-end measurements by up-to 150% for the average and 200% for the 99th percentile latency of Memcached. Motivated by the above, we provide

recommendations regarding the experimental environment configuration so that any unnecessary time bias is avoided and so that the results of the experiment reflect closely the behaviour of the target environment. These results support that the client-side configuration should be considered when designing experiments.

## ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their insightful comments on earlier versions of this manuscript. This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 101029391.

## REFERENCES

- [1] "Memcached: A Distributed Memory Object Caching System," online, accessed November 2021 <https://memcached.org/>.
- [2] A. Alameldeen and D. Wood, "Variability in architectural simulations of multi-threaded workloads," in *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.*, 2003, pp. 7–18.
- [3] G. Antoniou, H. Volos, D. B. Bartolini, T. Rollet, Y. Sazeides, and J. H. Yahya, "Agilepkgc: An agile system idle state architecture for energy proportional datacenter servers," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 851–867.
- [4] E. Asyabi, A. Bestavros, E. Sharafzadeh, and T. Zhu, "Peafowl: In-application CPU Scheduling to Reduce Power Consumption of In-memory Key-value Stores," in *SoCC*, 2020.
- [5] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," in *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 53–64. [Online]. Available: <https://doi.org/10.1145/2254756.2254766>
- [6] A. Belay, G. Prekas, M. Primorac, A. Klimovic, S. Grossman, C. Kozyrakis, and E. Bugnion, "The ix operating system: Combining low latency, high throughput, and efficiency in a protected dataplane," *ACM Trans. Comput. Syst.*, vol. 34, no. 4, dec 2016. [Online]. Available: <https://doi.org/10.1145/2997641>
- [7] C.-H. Chou, L. N. Bhuyan, and D. Wong, " $\mu$ DPM: Dynamic Power Management for the Microsecond Era," in *HPCA*, 2019.
- [8] C.-H. Chou, D. Wong, and L. N. Bhuyan, "Dynsleep: Fine-grained power management for a latency-critical data center application," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ser. ISLPED '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 212–217. [Online]. Available: <https://doi.org/10.1145/2934583.2934616>
- [9] Colin Ian King, "Stress-ng," online, accessed June 2024, Oct 21, <https://kernel.ubuntu.com/~cking/stress-ng/>.
- [10] M. Curiel and A. Pont, "Workload generators for web-based systems: Characteristics, current status, and challenges," *IEEE Communications Surveys and Tutorials*, vol. 20, no. 2, pp. 1526–1546, 2018.
- [11] D. Duplyakin, A. Uta, A. Maricq, and R. Ricci, "In datacenter performance, the only constant is change," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2020, pp. 370–379. [Online]. Available: <https://doi.ieeecomputersociety.org/10.1109/CCGrid49817.2020.00-56>
- [12] D. Duplyakin, N. Ramesh, C. Imburgia, H. F. A. Sheikh, S. Jain, P. Tekta, A. Maricq, G. Wong, and R. Ricci, "Avoiding the ordering trap in systems performance measurement," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. Boston, MA: USENIX Association, Jul. 2023, pp. 373–386. [Online]. Available: <https://www.usenix.org/conference/atc23/presentation/duplyakin>

- [13] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, "The design and operation of CloudLab," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, Jul. 2019, pp. 1–14. [Online]. Available: <https://www.flux.utah.edu/paper/duplyakin-atc19>
- [14] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathii, N. Katariki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvisky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, and C. Delimitrou, "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 3–18. [Online]. Available: <https://doi.org/10.1145/3297858.3304013>
- [15] A. Gendler, E. Knoll, and Y. Sazeides, "I-dvfs: Instantaneous frequency switch during dynamic voltage and frequency scaling," *IEEE Micro*, vol. 41, no. 5, pp. 76–84, 2021.
- [16] C. Gough, I. Steiner, and W. Saunders, *Energy Efficient Servers: Blueprints for Data Center Optimization*. Apress Berkeley, CA, 01 2015.
- [17] T. Hoefler and R. Belli, "Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results," in *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015, pp. 1–12.
- [18] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques For Experimental Design, Measurement, Simulation, and Modeling*. NY: Wiley, 04 1991.
- [19] James Lewis and Martin Fowler, "Microservices," online, accessed June 2024, March 2014, <https://martinfowler.com/articles/microservices.html>.
- [20] I. Jimenez, C. Maltzahn, J. Lofstead, A. Moody, K. Mohror, R. Arpacidusseau, and A. Arpacidusseau, "Characterizing and reducing cross-platform performance variability using os-level virtualization," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2016, pp. 1077–1080.
- [21] T. Kalibera, L. Bulej, and P. Tuma, "Benchmark Precision and Random Initial State," in *Proc. International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*. SCS, 2005, pp. 484–490.
- [22] H. Kasture and D. Sanchez, "Tailbench: a benchmark suite and evaluation methodology for latency-critical applications," in *2016 IEEE International Symposium on Workload Characterization (IISWC)*, 2016, pp. 1–10.
- [23] Kernel Development Team, "CPU Performance Scaling - The Linux Kernel documentation," online, accessed June 2024 <https://www.kernel.org/doc/html/v4.14/admin-guide/pm/cpufreq.html>.
- [24] M. Kogias, S. Mallon, and E. Bugnion, "Lancet: A self-correcting latency measuring tool," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. Renton, WA: USENIX Association, Jul. 2019, pp. 881–896. [Online]. Available: <https://www.usenix.org/conference/atc19/presentation/kogias-lancet>
- [25] J.-Y. Le Boudec, *Performance Evaluation of Computer and Communication Systems*. EPFL Press, 2010.
- [26] J. Leverich and C. Kozyrakis, "Reconciling high server utilization and sub-millisecond quality-of-service," in *Proceedings of the Ninth European Conference on Computer Systems*, ser. EuroSys '14. New York, NY, USA: Association for Computing Machinery, 2014. [Online]. Available: <https://doi.org/10.1145/2592798.2592821>
- [27] G. Liu, H. Zhang, M. Feng, L. Wong, and S.-K. Ng, "Supporting exploratory hypothesis testing and analysis," *ACM Trans. Knowl. Discov. Data*, vol. 9, no. 4, jun 2015. [Online]. Available: <https://doi.org/10.1145/2701430>
- [28] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu, "Characterizing microservice dependency and performance: Alibaba trace analysis," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 412–426. [Online]. Available: <https://doi.org/10.1145/3472883.3487003>
- [29] A. Maricq, D. Duplyakin, I. Jimenez, C. Maltzahn, R. Stutsman, and R. Ricci, "Taming performance variability," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 409–425. [Online]. Available: <https://www.usenix.org/conference/osdi18/presentation/maricq>
- [30] A. Mirhosseini, B. L. West, G. W. Blake, and T. F. Wenisch, "Qzilla: A scheduling framework and core microarchitecture for tail-tolerant microservices," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 207–219.
- [31] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney, "Producing wrong data without doing anything obviously wrong!" in *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XIV. New York, NY, USA: Association for Computing Machinery, 2009, p. 265–276. [Online]. Available: <https://doi.org/10.1145/1508244.1508275>
- [32] P. Nikolaou, Y. Sazeides, A. Lampropoulos, D. Guillhot, A. Bartoli, G. Papadimitriou, A. Chatzidimitriou, D. Gizopoulos, K. Tovletoglou, L. Mukhanov, and G. Karakonstantis, "On the evaluation of the total-cost-of-ownership trade-offs in edge vs cloud deployments: A wireless-denial-of-service case study," *IEEE Transactions on Sustainable Computing*, vol. 7, no. 2, pp. 334–345, 2022.
- [33] P. Nikolaou, Y. Sazeides, A. Lampropoulos, D. Guillhot, A. Bartoli, G. Papadimitriou, A. Chatzidimitriou, D. Gizopoulos, K. Tovletoglou, L. Mukhanov, G. Karakonstantis, M. Kleanthous, and A. Prat, *Total Cost of Ownership Perspective of Cloud vs Edge Deployments of IoT Applications*. Cham: Springer International Publishing, 2022, pp. 141–161. [Online]. Available: [https://doi.org/10.1007/978-3-030-74536-3\\_6](https://doi.org/10.1007/978-3-030-74536-3_6)
- [34] G. Prasad Srinivasa, S. Haseley, G. Challen, and M. Hempstead, "Quantifying process variations and its impacts on smartphones," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 117–126.
- [35] Rob Brigham, "DevOps at Amazon: A Look at Our Tools and Processes." online, accessed June 2024, <https://www.slideshare.net/AmazonWebServices/devops-at-amazon-a-look-at-our-tools-and-processes>.
- [36] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015. [Online]. Available: <https://networkrepository.com>
- [37] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965. [Online]. Available: <http://www.jstor.org/stable/2333709>
- [38] A. Sriraman and T. F. Wenisch, "μ Suite: A Benchmark Suite for Microservices," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*. Washington, DC, USA: IEEE, 2018, pp. 1–12.
- [39] A. Suresh and A. Gandhi, "Using variability as a guiding principle to reduce latency in web applications via os profiling," in *The World Wide Web Conference*, ser. WWW '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1759–1770. [Online]. Available: <https://doi.org/10.1145/3308558.3313406>
- [40] K. D. Team, "No\_hz: Reducing scheduling-clock ticks." online, accessed June 2024 [https://docs.kernel.org/timers/no\\_hz.html](https://docs.kernel.org/timers/no_hz.html).
- [41] D. Tsafirir, K. Ouaknine, and D. G. Feitelson, "Reducing performance evaluation sensitivity and variability by input shaking," in *2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2007, pp. 231–237.
- [42] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: maximizing on-chip parallelism," *SIGARCH Comput. Archit. News*, vol. 23, no. 2, p. 392–403, may 1995. [Online]. Available: <https://doi.org/10.1145/225830.224449>
- [43] Twitter, "Decomposing Twitter: Adventures in Service Oriented Architecture." online, accessed June 2024, [www.slideshare.net/InfoQ/decomposing-twitter-adventures-in-serviceoriented-architecture](http://www.slideshare.net/InfoQ/decomposing-twitter-adventures-in-serviceoriented-architecture).
- [44] A. Uta, A. Custura, D. Duplyakin, I. Jimenez, J. Rellermeier, C. Maltzahn, R. Ricci, and A. Iosup, "Is big data performance reproducible in modern cloud networks?" in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 513–527. [Online]. Available: <https://www.usenix.org/conference/nsdi20/presentation/uta>
- [45] C. Wang, K. Arya, M. Kogias, M. Vanga, A. Bhandari, N. J. Yadwadkar, and K. Schwan, "Statistical techniques for online anomaly detection in data centers," in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, 2011, pp. 385–392.
- [46] Y. Wang, K. Arya, M. Kogias, M. Vanga, A. Bhandari, N. J. Yadwadkar, S. Sen, S. Elnikety, C. Kozyrakis, and R. Bianchini, "Smartharvest: harvesting idle cpus safely and efficiently in the cloud," in *Proceedings of the Sixteenth European Conference on Computer Systems*, ser. EuroSys '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1–16. [Online]. Available: <https://doi.org/10.1145/3447786.3456225>

- [47] N. J. Wright, S. Smallen, C. M. Olschanowsky, J. Hayes, and A. Snively, "Measuring and understanding variation in benchmark performance," in *2009 DoD High Performance Computing Modernization Program Users Group Conference*, 2009, pp. 438–443.
- [48] J. H. Yahya, H. Volos, D. B. Bartolini, G. Antoniou, J. S. Kim, Z. Wang, K. Kalaitzidis, T. Rollet, Z. Chen, Y. Geng, O. Mutlu, and Y. Sazeides, "Agilewatts: An energy-efficient cpu core idle-state architecture for latency-sensitive server applications," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 835–850.
- [49] X. Zhan, R. Azimi, S. Kanev, D. Brooks, and S. Reda, "CARB: A C-state Power Management Arbiter for Latency-critical Workloads," *CAL*, 2016.
- [50] Y. Zhang, D. Meisner, J. Mars, and L. Tang, "Treadmill: attributing the source of tail latency through precise load testing and statistical inference," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. IEEE Press, 2016, p. 456–468. [Online]. Available: <https://doi.org/10.1109/ISCA.2016.47>
- [51] Y. Zhao, D. Duplyakin, R. Ricci, and A. Uta, "Cloud performance variability prediction," in *Companion of the ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 35–40. [Online]. Available: <https://doi.org/10.1145/3447545.3451182>
- [52] X. Zhou, X. Peng, T. Xie, J. Sun, C. Xu, C. Ji, and W. Zhao, "Benchmarking microservice systems for software engineering research," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, M. Chaudron, I. Crnkovic, M. Chechik, and M. Harman, Eds. ACM, 2018, pp. 323–324. [Online]. Available: <https://doi.org/10.1145/3183440.3194991>