A Study of Java Microbenchmark Tail Latencies

Sen He
Augusta University
Augusta, Georgia, USA
sehe@augusta.edu

In Kee Kim
University of Georgia
Athens, Georgia, USA
inkee.kim@uga.edu

Wei Wang
The University of Texas at San Antonio
San Antonio, Texas, USA
wei.wang@utsa.edu

ABSTRACT

As Java microbenchmarks are great at profiling the performance of essential code elements, they are widely adopted for Java performance testing. Performance testing using Java microbenchmarks is composed of two phases: the warmup phase and the steady phase. Usually, testing results from the warmup phase are discarded because of the highly fluctuating performance caused by the optimization of Java Virtual Machine. The performance results collected during the steady phase are used for performance evaluation as they are assumed to be more stable. However, according to our study, *severe performance fluctuations also occur during the steady phase, which leads to long tail latencies.* Long tail latencies constitute a big problem in modern Java systems (of course, other systems and applications) as they hurt the user experience by prolonging the overall execution time.

In this paper, we extensively evaluated the long tail performance of 586 Java microbenchmarks from 30 Java systems. The evaluation results show that, for 38% of the benchmarks in steady phase, their 99%tile execution times are over 30% higher than their median execution times. In the worst-case scenario, the 99%ile performance is 659 times higher than the median performance. Furthermore, the 95%ile execution times are above 30% higher than the median execution times for 11% of the steady phase benchmarks.

CCS CONCEPTS

• **General and reference** → *Measurement*; **Performance**; • **Software and its engineering** → **Software performance**.

KEYWORDS

Tail Latency; Performance Evaluation; Java; Microbenchmark

ACM Reference Format:
Sen He, In Kee Kim, and Wei Wang. 2023. A Study of Java Microbenchmark Tail Latencies. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23 Companion), April 15–19, 2023, Coimbra, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3578245.3584690

ICPE '23 Companion, April 15–19, 2023, Coimbra, Portugal

ACM ISBN 979-8-4007-0072-9/23/04…$15.00
https://doi.org/10.1145/3578245.3584690

1 INTRODUCTION

A common practice to determine the performance of any application is performance testing [2, 18]. Typically, performance testing is carried out by repeatedly running the application-under-test until enough performance data points are collected [8, 9]. Applications used for performance testing can be macrobenchmarks or microbenchmarks. Macrobenchmarks are comprehensive tests that measure the overall performance of a system by simulating real-world workloads and are designed to provide a broad measure of system performance. Microbenchmarks, on the other hand, are small, isolated test cases used to evaluate the performance of a specific piece of code or function. [5] Microbenchmarks are increasingly popular and widely adopted mainly due to their short runtimes and testing durations [10, 12].

Specifically, microbenchmarks are broadly used to test the performance of Java applications [14]. Java microbenchmarking is divided into two phases: the warmup phase and the steady phase. In the warmup phase, the testing results and performance of Java benchmarks often show high variability due to the Java Virtual Machine (JVM) optimizations [1, 17]. Thus, the testing results collected during the warmup phase tend to be discarded. After entering the steady phase, ideally, the performance should be more stable, and the testing results from the steady phase can be used for tests and analyses to understand the performance of Java applications [1, 17].

However, our study reveals that even during the steady phase, severe performance fluctuations also occur so that such fluctuations lead to long tail latencies. Tail latencies are important because they measure the performance of a small percentage of requests that take considerably longer to complete than the other requests. Long tail latencies can greatly affect the user experience and the overall performance of a system [3, 7, 15]. In addition, tail latencies can indicate the performance bottlenecks that need to be addressed [4, 16]. In this paper, we extensively evaluated the long tail performance of 586 Java microbenchmarks from 30 Java systems [17]. For those 521 Java microbenchmarks that can reach the steady state, we found that 38% of them have long tail latencies. *i.e.*, 99%ile execution times are over 30% higher than their median execution times.

The contributions of this paper are as follows: First, we extensively evaluated the tail performance on a large dataset of Java microbenchmarks, and we found that the performance still severely fluctuated during the steady stage. Second, we found that when studying tail latencies for Java microbenchmarks, including the performance data from both the warmup and steady phases will not change the tail performance significantly. Finally, we showed that using median to assist tail latency analysis is better than using mean performance, because the median performance is less affected by extreme performance values like the tail performance.

Table 1: Benchmark suites used in the evaluation, including the GitHub organization *(GitHub Org.)*, the name of the benchmark suite *(Bench. Name)*, and the number of benchmarks inside each suite *(Num.)*

GitHub Org.	Bench. Name	Num.
apache	arrow	46
raphw	byte-buddy	39
apache	camel	35
cantaloupe-project	cantaloupe	103
prometheus	client_java	33
crate	crate	39
eclipse	eclipse-collections	2415
h2oai	h2o-3	73
hazelcast	hazelcast	144
HdrHistogram	HdrHistogram	75
apache	hive	1402
imglib	imglib2	25
JCTools	JCTools	172
jdbi	jdbi	76
eclipse	jetty.project	212
jgrapht	jgrapht	91
apache	kafka	3578
zalando	logbook	20
apache	logging-log4j2	572
netty	netty	1746
prestodb	presto	1534
protostuff	protostuff	31
r2dbc	r2dbc-h2	20
eclipse	rdf4j	132
RoaringBitmap	RoaringBitmap	1620
ReactiveX	RxJava	1302
yellowstonegames	SquidLib	334
apache	tinkerpop	57
eclipse-vertx	vert.x	41
openzipkin	zipkin	63

2 TAIL LATENCY TESTING METRICS

Long tail latencies from code snippets can significantly worsen a system's overall performance, ultimately leading to poor user experiences. To evaluate the tail performance, we use the percentage difference $diff$ to quantify the performance impacts caused by long tail latencies. For each microbenchmark, the percentage difference $diff$ is calculated by comparing the tail performance $Perf_{tail}$ with the median performance, $Perf_{median}$, using the following equation,

$$diff = |\frac{Perf_{tail} - Perf_{median}}{Perf_{median}}| \times 100\%. \quad (1)$$

As described in Section 1, tail latencies are measured as the performance of a small percentage of requests that take considerably longer to complete than the other requests. Ideally, the percentage difference $diff$ should be calculated by comparing $Perf_{tail}$ with the mean performance $Perf_{mean}$. However, we selected the median performance over the mean performance because the median is a more robust statistic, indicating that it is less affected by changes and outliers in the given dataset. In particular, the median performance is chosen over the mean performance because it is less affected by extreme performance values like the tail performance.

3 EVALUATION

3.1 Evaluation Setup.

This section describes the dataset and experimental evaluations we adopted for this study.

Benchmarks and Environments. We used the benchmark datasets collected by Traini et al. [17] for the evaluation. The datasets provide the performance results of 586 microbenchmarks from 30 Java benchmark suites. Those 30 Java benchmark suites (shown in Table 1) were selected based on their popularity on GitHub. From these benchmark suites, 586 benchmarks were randomly picked for evaluation. For each benchmark, 10 JHM (Java Microbenchmark Harness) forks were executed, and each fork contained 3000 benchmark invocations for at least 300 seconds of execution time [17]. All the benchmarks were tested on a bare-metal server with 40 cores (dual 2.3GHz Intel Xeon E5-2650v3 CPU) and 80 GiB of RAM running Ubuntu Linux 18.04.2 LTS. To reduce the potential performance-affecting factors, Traini et al. disabled Intel Turbo Boost, Address Space Layout Randomization, unnecessary Linux processes and daemons, and SSH login. They also fixed the available JVM heap memory to 8GB and reduced context switching [17].

Evaluation Methodologies. The relative performance deviation for non-steady forks can be significantly reduced by continuously warming each fork up for 300 iterations [17]. For both steady and non-steady forks, it's safe to assume that the data points collected after 1500 iterations have relatively low performance fluctuations caused by JVM optimizations. Thus, we can use the data points collected after 1500 iterations for evaluation. To split the forks that never entered the steady phase, we adopted the approach of Kalibera et al. [11], and the parameters from Traini et al. [17].

For the first set of evaluations, we tested the tail latencies using all the data points collected after 1500 iterations; that is, we used 1500 data points from iteration-1500 to iteration-3000 to test each steady-phase fork. In the second set of evaluations, we reduced the sample size to 500 (from iteration-2500 to iteration-3000) for each steady-phase fork to see how the tail latency analysis would be affected by different sample sizes. In the last set of evaluations, to check if including data points from the warmup phase impacts the tail latency analysis, we adopted all data points from both steady and non-steady phases.

Parameters. In this evaluation, we chose 95%ile performance (P95) and 99%ile performance (P99) to represent the tail latencies. Then, we can calculate the percentage difference $diff$ using the equation in Section 2. After the $diff$ were calculated, we compared to see the percentages of $diff$ that are greater than 5%, 30%, and 50%.

3.2 Experiment Results and Discussion

In this subsection, we present and discuss the long tail performance observed from the benchmark executions. This evaluation seeks to answer the following **research questions**: 1) How severe are tail latencies in Java microbenchmarks? 2) Will data points from warmup phases affect tail latency analysis?

Due to space limitations, we only show the graph of four benchmarks, including jdbi[1], hdr[2], bytebuddy[3], and apache[4].

P99 Performance. Figure 1 presents the percentage difference $diff_{99\%ile}$ between P99 execution time and median execution time for four benchmarks. Specifically, each benchmark contains 10 forks,

[1] org.jdbi.v3.benchmark.QualifiersBenchmark.mapUnqualifiedBean
[2] bench.HdrHistogramEncodingBench.skinnyEncodeIntoCompressedByteBuffer
[3] net.bytebuddy.benchmark.TrivialClassCreationBenchmark.benchmarkJdkProxy
[4] org.apache.kafka.jmh.record.RecordBatchIterationBenchmark

and the $diff_{99\%ile}$ for each fork is calculated from 1500 data points. As the figure shows, all four benchmarks have long tail latencies. For example, in Figure 3a, the sixth fork (Fork6) of jdbi benchmark holds a $diff_{99\%ile}$ value of 658.7, which means the P99 execution time is 65870% higher than the median execution time. Similarly, the $diff_{99\%ile}$ for hdr, bytebuddy, and apache in the worst-case scenario are 4670%, 1860%, and 1340%, respectively. Those results demonstrated that tail latencies could negatively affect the overall performance and user experiences.

Additionally, from the figure, we conclude that the tail latencies fluctuate severely. For instance, the $diff_{99\%ile}$ for the 10 forks of jdbi ranges from 290% to 65870%. For hdr, bytebuddy, and apache, the $diff_{99\%ile}$ varies from 3140% to 4670%, 330% to 1860%, and 890% to 1340%, separately. The highly fluctuating testing results suggested that to accurately obtain the tail performance, more forks are required for each Java microbenchmark. *A new testing methodology is demanded to determine the number of forks to get highly accurate tail performance.*

Figure 2 shows the percentage difference for the same four benchmarks when each of the $diff_{99\%ile}$ is calculated from 500 data points. The figure delivers very similar information as Figure 1 that: a). Long tail latencies exist and could severely affect the overall performance. b). The tail performance highly fluctuates. Specifically, when comparing the results from Figures 1 and 2, the fluctuation range and trend of $diff_{99\%ile}$ appear to have similar patterns. This result indicates that extra test runs for each fork will not benefit tail latency analysis when the number of test runs for each fork is over a certain threshold. *A new testing methodology is required to decide the threshold for each Java microbenchmark to reduce undue test runs while maintaining the accuracy of the tail performance results.* It is worth noting that the worst-case scenario in Figure 2a (315020% difference in Fork1) is caused by the low median results of Fork1, and it simply implies that more forks are required to be tested.

The evaluation reveals that tail latencies exist in most of the benchmarks. For all the benchmark forks (tested with 1500 data points) that can enter the steady phase, 77.1% of them have a $diff_{99\%ile}$ value greater than 5%, 38.4% of them have a $diff_{99\%ile}$ value greater than 30%, and 21% of them have a $diff_{99\%ile}$ value greater than 50%. Also, for all the steady-phase benchmark forks (tested with 500 data points), 74.3% of them have a $diff_{99\%ile}$ value greater than 5%, 36.6% of them have a $diff_{99\%ile}$ value greater than 30%, and 20% of them have a $diff_{99\%ile}$ value greater than 50%.

P95 Performance. Figures 3 and 4 show the percentage difference $diff_{95\%ile}$ between P95 and median performance. Each percentage difference in Figure 3 is calculated from evaluation set1 (1500 data points), while each $diff_{95\%ile}$ in Figure 4 is computed from evaluation set2 (500 data points). Comparing to Figure 1 and Figure 2, $diff_{95\%ile}$ have smaller fluctuation range than $diff_{99\%ile}$. However, the P95 performance is still high. For all the benchmark forks in the steady phase (evaluation set1), 62.1% of them have a $diff_{95\%ile}$ value greater than 5%, 21.3% of them have a $diff_{95\%ile}$ value greater than 30%, and 7.6% of them have a $diff_{95\%ile}$ value greater than 50%. For all the steady-phase benchmark forks (evaluation set2), 55.8% of them have a $diff_{95\%ile}$ value greater than 5%, 19.2% of them have a $diff_{95\%ile}$ value greater than 30%, and 7.4% of them have a $diff_{95\%ile}$ value greater than 50%.

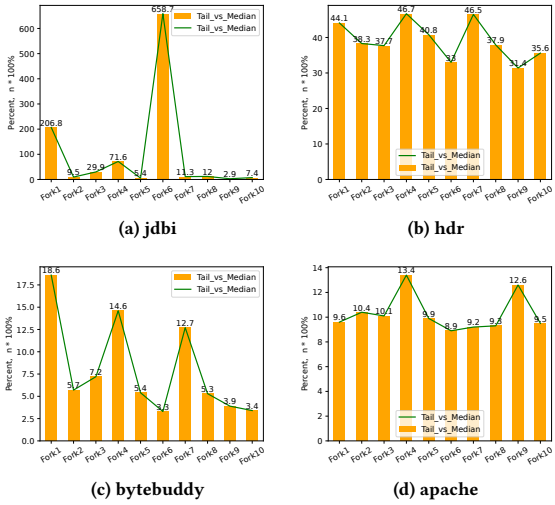


Figure 1: The percentage difference between P99 and median performance, tested with 1500 data points for each fork.

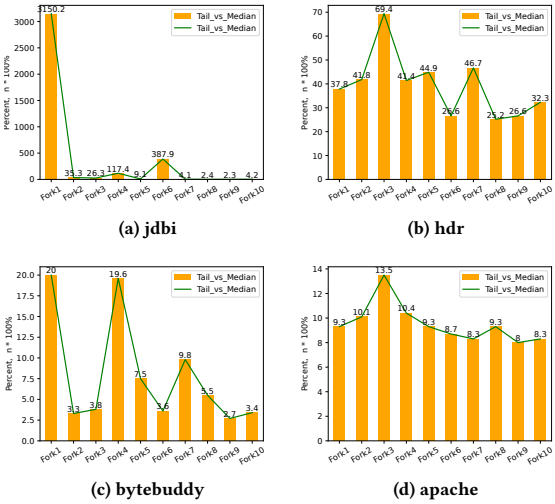
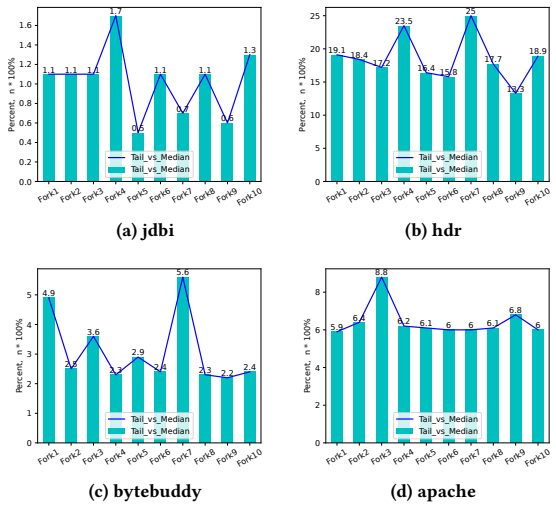


Figure 2: The percentage difference between P99 and median performance, tested with 500 data points for each fork.

Tail Latency Analysis Using Data from Both Warmup and Steady Phases. As discussed in Section 1, Java performance testing data collected during the warmup phase (non-steady phase) is usually discarded [1, 6, 11]. However, in Figure 5, for all four benchmarks, the fluctuation scopes of P99 performance computed from 500 data points, 1500 data points, and 3000 data points (all data) are very close to each other. On the other hand, the fluctuation scopes of P99 performance calculated by 50 data points are signifcantly different, especially for jdbi, hdr, and apache.

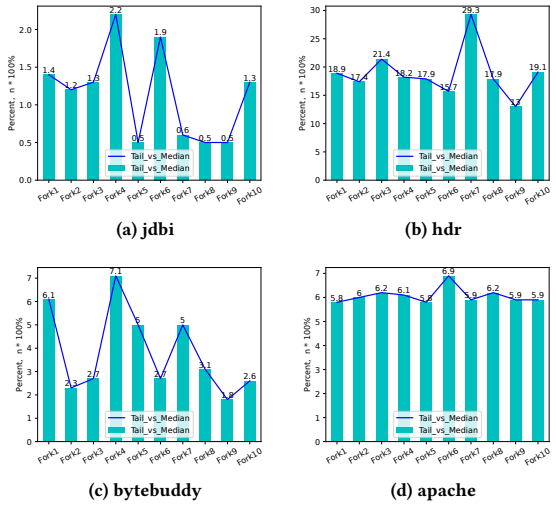
Furthermore, using data points from both the warmup and the steady phases (evaluation set3, 3000 data points), we calculated the $diff_{99\%ile}$ for all benchmark forks that can eventually enter the



(a) jdbi (b) hdr

(c) bytebuddy (d) apache

Figure 3: The percentage difference between P95 and median performance, tested with 1500 data points for each fork.



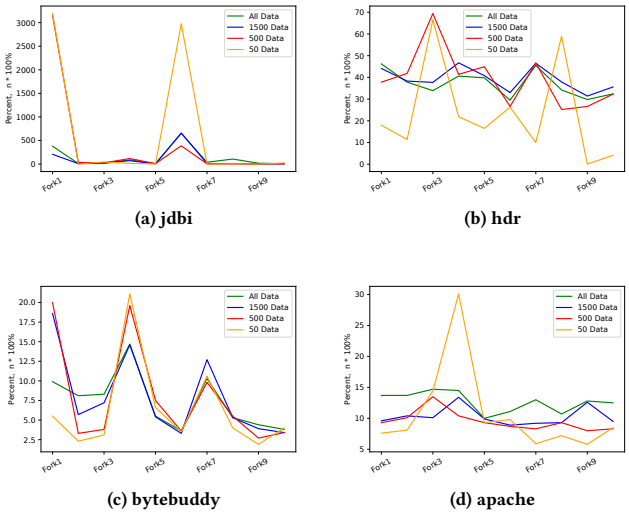
(a) jdbi (b) hdr

(c) bytebuddy (d) apache

Figure 4: The percentage difference between P95 and median performance, tested with 500 data points for each fork.

steady stage. For P99, 80.4% of them have a $dif f_{99\%ile}$ value greater than 5%, 42.6% of them have a $dif f_{99\%ile}$ value greater than 30%, and 24.9% of them have a $dif f_{99\%ile}$ value greater than 50%. The results are very similar to those from the evaluation set1, which illustrate that: *including the performance data from both the warmup and steady phases will not change the tail performance significantly if the overall testing length is sufficient.*

Using Mean Performance vs. Median Performance. Mean performance can be a useful metric for determining how well a Java microbenchmark performs. However, when evaluating the tail latencies, using mean performance may introduce unnecessary fluctuations to the results. For example, in Figure 1d and Figure 2d, the



(a) jdbi (b) hdr

(c) bytebuddy (d) apache

Figure 5: The percentage difference between P99 and median performance, using data from both the warmup (non-steady) phase and the steady phase, tested with 50, 500, 1500, and all data points, respectively.

mean performance differences ($Comp_{mean} = |\frac{Mean_{evalset1} - Mean_{evalset2}}{Mean_{evalset1}}|$) between evaluation set1 and evaluation set 2 for the 10 forks ranges from 0.1% to 11.0%. On the other hand, the median performance differences ($Comp_{mean} = |\frac{Median_{evalset1} - Median_{evalset2}}{Median_{evalset1}}|$) only ranges from 0.1% to 0.7%. Less performance fluctuation in the median is because of its robustness to outliers and/or abnormal results, meaning that the median is also less affected by the tail latency. Therefore, when analyzing the tail performances, choosing median performance over mean performance is more desirable and can minimize the impact of data uncertainties.

4 CONCLUSION AND FUTURE WORK

In this study, we extensively evaluated the tail latencies on a large dataset of Java microbenchmarks from Traini et. al [17], which contains 586 Java microbenchmarks from 30 popular Java systems. We observed that, during the steady phase, the performance still severely fluctuates, and the tail performance is considerably higher than the median performance. Specifically, our analysis showed that over 38% of the steady-phase benchmark forks, their 99%ile latencies are above 30% higher than their median latencies. Performance fluctuations over 30% can negatively affect the overall system performance and user experiences. Based on the analysis, we conclude that using the data from both the warmup and steady phases to test the tail performance will not change the testing results significantly if the overall testing period is long enough.

To accurately obtain the tail performance of Java microbenchmarks, more testing forks are required. This leaves us with the first future work: "To develop a new testing methodology which can assist user determining the number of forks to get highly accurate tail performance." Additionally, to reduce the testing time and cost while maintaining high accuracy, our second future work is to improve existing dynamic reconfiguration techniques [13].

REFERENCES

[1] Edd Barrett, Carl Friedrich Bolz-Tereick, Rebecca Killick, Sarah Mount, and Laurence Tratt. 2017. Virtual Machine Warmup Blows Hot and Cold. *Proc. ACM Program. Lang.* 1, OOPSLA, Article 52 (oct 2017), 27 pages. https://doi.org/10.1145/3133876

[2] Andreas Burger, Heiko Koziolek, Julius Rückert, Marie Platenius-Mohr, and Gösta Stomberg. 2019. Bottleneck Identification and Performance Modeling of OPC UA Communication Models. In *Proc. of ACM/SPEC Int'l Conf. on Performance Engineering (ICPE '19)*.

[3] Wenzhi Cui, Daniel Richins, Yuhao Zhu, and Vijay Janapa Reddi. April 14, 2019. Tail latency in node.js: energy efficient turbo boosting for long latency requests in event-driven web services. In *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'19)*. Providence, RI, USA.

[4] Jeffrey Dean and Luiz André Barroso. 2013. The Tail at Scale. *Commun. ACM* 56, 2 (feb 2013), 74–80. https://doi.org/10.1145/2408776.2408794

[5] Written by Douglas Eadline. [n.d.]. Douglas Eadline. https://www.clustermonkey.net/Benchmarking-Methods/micro-benchmarks-vs-macro-benchmarks.html

[6] Andy Georges, Dries Buytaert, and Lieven Eeckhout. 2007. Statistically Rigorous Java Performance Evaluation. *SIGPLAN Not.* 42, 10 (oct 2007), 57–76. https://doi.org/10.1145/1297105.1297033

[7] Hao He, Runzhi He, Haiqiao Gu, and Minghui Zhou. August 23-28, 2021. A large-scale empirical study on Java library migrations: prevalence, trends, and rationales. In *29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'21)*. Athens, Greece.

[8] Sen He, Tianyi Liu, Palden Lama, Jaewoo Lee, In Kee Kim, and Wei Wang. 2021. Performance Testing for Cloud Computing with Dependent Data Bootstrapping. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 666–678. https://doi.org/10.1109/ASE51524.2021.9678687

[9] Sen He, Glenna Manns, John Saunders, Wei Wang, Lori Pollock, and Mary Lou Soffa. 2019. A Statistics-Based Performance Testing Methodology for Cloud Applications. In *Proc. of ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering*.

[10] Zhen Ming Jiang and Ahmed E. Hassan. 2015. A Survey on Load Testing of Large-Scale Software Systems. *IEEE Transactions on Software Engineering* 41, 11 (2015), 1091–1118. https://doi.org/10.1109/TSE.2015.2445340

[11] Tomas Kalibera and Richard Jones. 2013. Rigorous Benchmarking in Reasonable Time. In *Proceedings of the 2013 International Symposium on Memory Management* (Seattle, Washington, USA) *(ISMM '13)*. Association for Computing Machinery, New York, NY, USA, 63–74. https://doi.org/10.1145/2491894.2464160

[12] Christoph Laaber and Philipp Leitner. 2018. An Evaluation of Open-Source Software Microbenchmark Suites for Continuous Performance Assessment. In *Proceedings of the 15th International Conference on Mining Software Repositories* (Gothenburg, Sweden) *(MSR '18)*. Association for Computing Machinery, New York, NY, USA, 119–130. https://doi.org/10.1145/3196398.3196407

[13] Christoph Laaber, Stefan Würsten, Harald C. Gall, and Philipp Leitner. 2020. Dynamically Reconfiguring Software Microbenchmarks: Reducing Execution Time without Sacrificing Result Quality. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 989–1001. https://doi.org/10.1145/3368089.3409683

[14] Philipp Leitner and Cor-Paul Bezemer. 2017. An Exploratory Study of the State of Practice of Performance Testing in Java-Based Open Source Projects. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering* (L'Aquila, Italy) *(ICPE '17)*. Association for Computing Machinery, New York, NY, USA, 373–384. https://doi.org/10.1145/3030207.3030213

[15] Waleed Reda, Marco Canini, P. Lalith Suresh, Dejan Kostic, and Sean Braithwaite. April 23-26, 2017. Rein: Taming Tail Latency in Key-Value Stores via Multiget Scheduling. In *ACM European Conference on Computer Systems (EuroSys'17)*. Belgrade, Serbia.

[16] Pasindu Tennage, Srinath Perera, Malith Jayasinghe, and Sanath Jayasena. 2019. An Analysis of Holistic Tail Latency Behaviors of Java Microservices. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. 697–705. https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00104

[17] Luca Traini, Vittorio Cortellessa, Daniele Di Pompeo, and Michele Tucci. 2023. Towards Effective Assessment of Steady State Performance in Java Software: Are We There Yet? *Empirical Softw. Engg.* 28, 1 (jan 2023), 57 pages. https://doi.org/10.1007/s10664-022-10247-x

[18] Yutong Zhao, Lu Xiao, Xiao Wang, Bihuan Chen, and Yang Liu. 2019. Localized or Architectural: An Empirical Study of Performance Issues Dichotomy. In *Int'l Conf. on Software Engineering: Companion Proceedings*.