About Benchmarks, Garbage Collectors & (Phantom) Latencies

Tail latencies of database systems turned out to be a serious issue for vendors of digital services as they can impair the overall user experience significantly. A group of researchers around Stefanie Scherzinger (University of Passau) showed, that the naive use of benchmarking tools may hamper the analysis of those latencies additionally. In the following I will give a brief insight into their experimental setup and results, based on the lecture „Tail Latencies in the DB/OS Stack“[1] and the related paper[2].

Research Question

„Tail latencies“ are a small share of a database system’s response times that are dramatically higher compared to the majority of all response times[3]- and thus form the „tail“ of a latency distribution plot. As these latencies especially will affect frequent users (who are probably the most important customers), providers of digital services (e.g. DBaaS) would do well to eliminate them. Undoubtedly, measuring the database performance in order to locate potential sources of delay is a first step to achieve this. The common java-based benchmarking framework OLTPBench[4] seems to provide a quick and easy help on this matter. However, it is reasonable to suppose, that it may negatively affect the accuracy of measurements, since the Java Virtual Machine’s garbage collection produces indeterministic stop-the-world-latencies. In order to verify this assumption, Scherzinger et al. examined the impact of different Java Virtual Machines and garbage collectors on measuring a database system’s latencies by the use of OLTPBench.

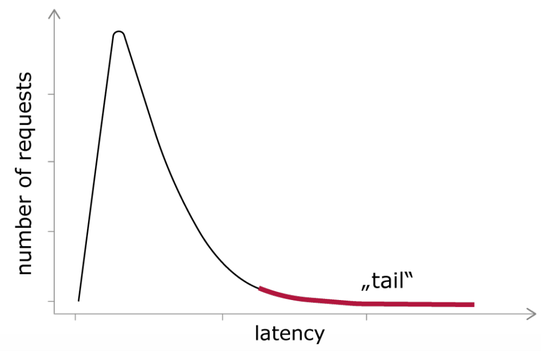


Figure inspired by [8].

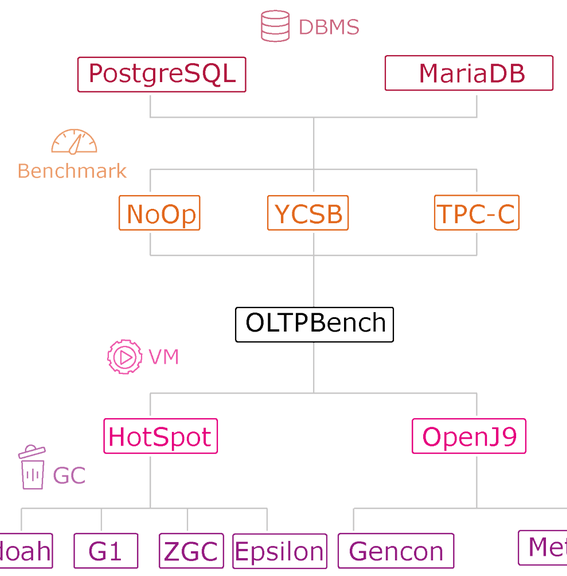
Methodology

Scherzinger et al. conducted a series of latency measurements (10 sec. warm-up phase, 60 sec. measuring), whereby they used various combinations of Java VM(executing OLTPBench), Garbage Collector (GC), benchmark and DBMS (see figure). Besides the requests’ latencies recorded by OLTPBench they also took the JVM log into account for their analysis since it gives information about latencies of GC events(see following section).

The TPC-C benchmark produces complex transactions simulating the workload of a wholesale supplier’s database system[5], whereas the transactions generated by the YCSB benchmark are rather lightweight. The NoOp Benchmark marks the minimum roundtrip time between database and client as it merely sends empty statements that do not evoke productive work inside the database.

The GCs used follow different design goals, for example, the G1 aiming both at low latency and high throughput[6], or the Z Garbage Collector focusing on low latency. One should also mention the special role of the Epsilon GC, which, as opposed to the other GCs, does not reclaim memory and thus minimizes latency overhead[7].

In order to prevent distortions caused on hardware level, they took several steps like disabling Simultaneous Multithreading, pinning benchmark and database processes to CPUs on the same NUMA-Node and turning off Turbo Boost® for a constant core frequency.



Different combinations of DBMS, Benchmark, VM and GC. Each path from bottom to top (or vice versa) represents a setup used for the experiments.

Main Results

(as the results turned out to be not depending on the DMBS, all results shown refer to MariaDB)

NoOp:

While for Epsilon (baseline) they observed that the highest latencies are mostly about 1 ms, the setup with G1 generates extreme values beyond 10 ms. As the grey dots represent the extreme values measured by the benchmark harness and the black dots visualize GC-latencies from JVM logs, it is reasonable to suppose, that these tail latencies are caused by the GC. In terms of gencon, by overlaying OLTPBench-measured latencies and latencies from JVM logs one can also discover how the GC affects the tail of the latency distribution.

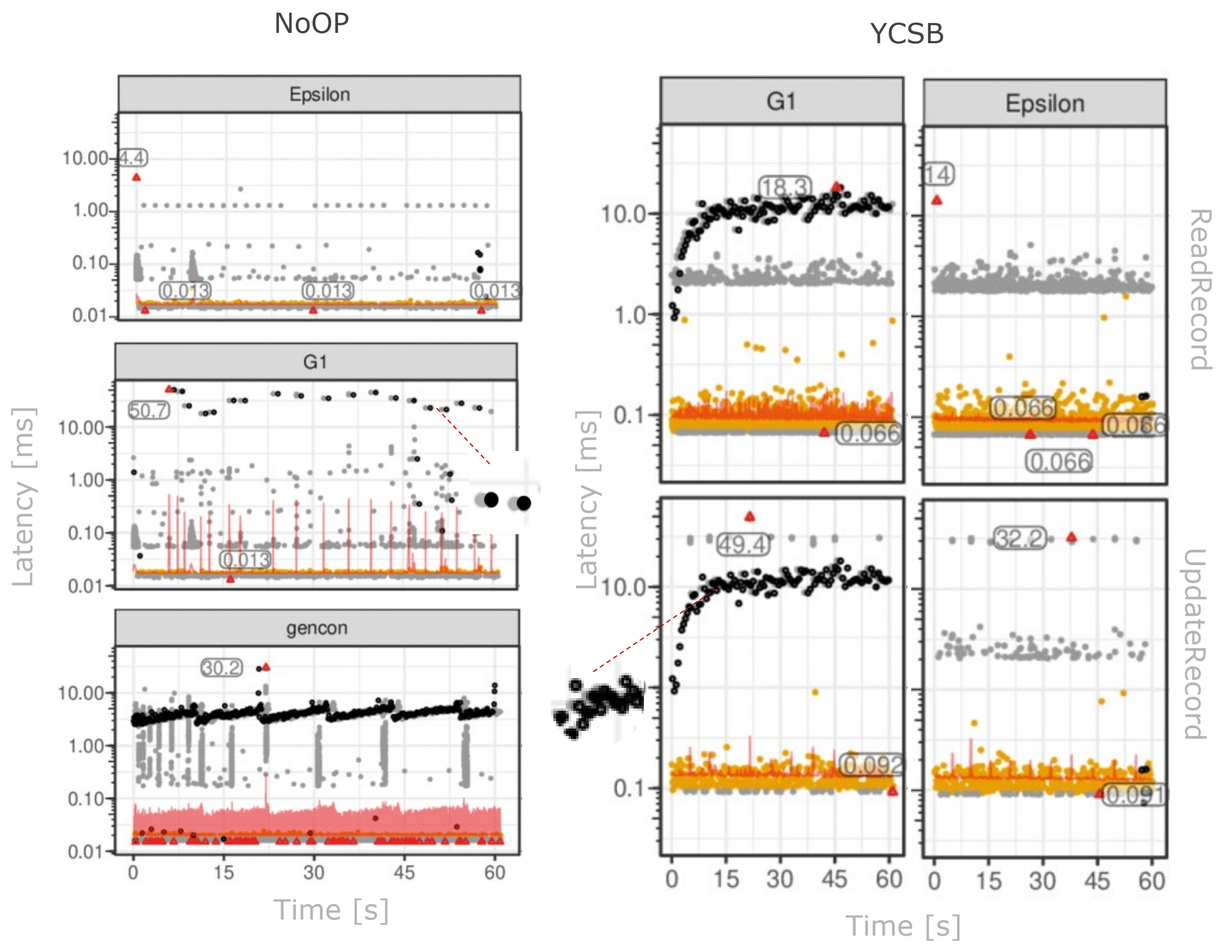
YCSB:

The measurement results for setups with YCSB are visualized separately for read and write transactions (ReadRecord and UpdateRecord). Just as for the setup with NoOp benchmark, the G1 GC has a significant impact on tail latencies, both for read and write transactions.

TCP-C:

As for setups including TPC-C, there is, no matter which GC is used, no considerable impact on the measurements to report. This can be explained by the fact, that TPC-C evokes a comparably high workload on database side, so that the benchmark overhead becomes insignificant.

To sum it up, the execution environment of OLTPBench can indeed have a distorting effect on measuring tail latencies - especially for setups using a benchmark producing rather small workloads on database side. Therefore, Scherzinger et al. encourage engineers and researchers to take this issue more seriously in future.



Plotted latencies for NoOP and YCSB.
Ochre dots: standard values (between 0.025th and 99.975th latency), down-sampled .
Grey dots: extreme Values.
Red Triangles: minimum and maximum latency.
Black dots: latencies from JVM Log. All diagrams taken from[2].

Katharina Hasenlust
Bachelor Student
Hasso Plattner Institute, Potsdam, Germany

E-Mail: katharina.hasenlust@hpi.de

[1] https://www.tele-task.de/lecture/video/8848/
[2] Fruth, Michael, Scherzinger, Stefanie, Mauerer, Wolfgang and Ramsauer, Ralf: Tell-Tale Tail Latencies: Pitfalls and Perils in Database Benchmarking. 2021.
[3] https://www.computerweekly.com/opinion/Storage-How-tail-latency-impacts-customer-facing-applications
[4] https://github.com/oltpbenchmark/oltpbench
[5] http://www.tpc.org/tpcc/
[6] https://www.oracle.com/technetwork/tutorials/tutorials-1876574.html
[7] https://openjdk.java.net/jeps/318
[8] https://files.speakerdeck.com/presentations/9c3de237cd08416b86d0599992ed7e7d/slide_27.jpg

Hasso Plattner Institut