Performance Interference on Key-Value Stores in Multi-tenant Environments: When Block Size and Write Requests Matter

Adriano Lange
alange@inf.ufpr.br
Federal Univeristy of Paraná (UFPR)
Curitiba, Brazil

Tiago Rodrigo Kepe
tiago.kepe@ifpr.edu.br
Federal Institute of Paraná (IFPR)
Curitiba, Brazil

Marcos Sfair Sunyé
sunye@inf.ufpr.br
Federal Univeristy of Paraná (UFPR)
Curitiba, Brazil

ABSTRACT

Key-value stores are currently used by major cloud computing vendors, such as Google, Facebook, and LinkedIn, to support large-scale applications with concurrent read and write operations. Based on very simple data access APIs, the key-value stores can deliver outstanding throughput, which have been hooked up to high-performance solid-state drives (SSDs) to boost this performance even further. However, measuring performance interference on SSDs while sharing cloud computing resources is complex and not well covered by current benchmarks and tools. Different applications can access these resources concurrently until becoming overloaded without notice either by the benchmark or the cloud application. In this paper, we define a methodology to measure the problem of performance interference. Depending on the block size and the proportion of concurrent write operations, we show how a key-value store may quickly degrade throughput until becoming almost inoperative while sharing persistent storage resources with other tenants.

CCS CONCEPTS

• **Information systems → Database performance evaluation**.

KEYWORDS

performance interference, key-value store, multi-tenant, flash disks

ACM Reference Format:
Adriano Lange, Tiago Rodrigo Kepe, and Marcos Sfair Sunyé. 2021. Performance Interference on Key-Value Stores in Multi-tenant Environments: When Block Size and Write Requests Matter. In *Companion of the 2021 ACM/SPEC International Conference on Performance Engineering (ICPE '21 Companion), April 19–23, 2021, Virtual Event, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3447545.3451191

1 INTRODUCTION

In this paper, we consider the performance interference of key-value stores when competing for persistent storage resources. In cloud computing environments, a common strategy to reduce costs is to increase tenants' density in the same hardware. However, this strategy comes with the inconvenience of performance interference among tenants due to the competition for computing resources. Even in a single-node, this performance interference may potentially impair an entire database cluster.

Benchmarking is a powerful tool to assess performance before deploying a system into production. Although YCSB [7] and db_bench [13] are representative key-value benchmarks, their major drawback is that they only analyze a one-sided perspective, without considering the multi-tenant effect and the underlying hardware.

In this paper, we design a benchmarking strategy, named "pressure scale," to uncover performance interference by putting pressure into the key-value stores' components. This strategy consists of a systematic methodology that progressively increases the concurrent workload intensity on a shared persistent storage device to reveal unpredictable performance states.

The class of storage devices considered by our benchmarking strategy corresponds to the solid-state drives (SSDs) based on flash memory. SSDs are becoming very common within cloud computing infrastructures. Despite the promised superior performance in many workloads, the internal architecture of these devices is far more complex than traditional hard disk drives (HDDs), i.e., the out-of-place updates due to the necessity of erasing blocks before writing and the use of flash translation layers (FTLs) [14]. Such a complexity not only hinders the performance modeling of a single application but also turns it even more complex in multi-tenant environments.

As the key-value store, we evaluate the performance of RocksDB [13], a storage engine based on Log-Structured Merge-trees (LSM-trees) that has its design focused on SSDs. RocksDB is currently used in production by many companies and cloud applications, as at Facebook, LinkedIn, and Yahoo.

By applying our methodology, we quantify the pressure of different concurrent workloads on the key-value store's performance. The experiments conducted in this work demonstrate that the performance impact is highly dependent on block size and the proportion of write operations in the concurrent workload mix. These results show that such a strategy is efficient in catching uncovered functional issues in RocksDB.

The remainder of this paper is organized as follows. Section 2 introduces the pressure scale methodology. Section 3 describes the experimental setup and shows the main results using the benchmarks YCSB and db_bench. Section 4 lists the related work and Section 5 summarizes the conclusions of this work.

2 THE PRESSURE SCALE METHODOLOGY

We propose the **pressure scale** methodology to measure the performance interference that a key-value storage engine may experience when sharing persistent storage resources with other concurrent workloads. The following definitions formalize the concepts used in this paper:

ICPE '21 Companion, April 19–23, 2021, Virtual Event, France

ACM ISBN 978-1-4503-8331-8/21/04…$15.00
https://doi.org/10.1145/3447545.3451191

Definition 1. *Let W be the set of possible concurrent workloads, and C be the set of performance criteria to be analyzed for a particular system. We define* ***pressure*** *$\rho : W \rightarrow C$ as a functional mapping between W and C, i.e., workloads that pressure the system performance.*

Definition 2. *Let $(C, \preceq_C)$ be a linearly ordered set, where $c_a \preceq_C c_b$ means "c_a is an inferior performance value than or equal to c_b" for c_a and $c_b \in C$. We denote* ***pressure scale*** *as a linear preorder $(W, \succeq_W)$, such that $w_a \succeq_W w_b$ iff $\rho(w_a) \preceq_C \rho(w_b)$. The relation $w_a \succeq_W w_b$ means "w_a exerts more or the same pressure than w_b."*

According to Definition 1, this paper considers the key-value store's throughput as the primary performance criterion (C). Once fewer throughput represents worse performance, $\preceq_C$ can be trivially defined as $\leq$ (Def. 2). We determine the pressure function (ρ) experimentally by measuring the key-value store's average performance for each instance of W. Considering w_0 as the condition with no concurrent workloads, we define **normalized pressure** as the difference between $\rho(w_0)$ and $\rho(w_i) \in W$ normalized by $\rho(w_0)$, i.e., $\frac{\rho(w_0)-\rho(w_i)}{\rho(w_0)}$.

Our objective is to define several instances of W that produce different degrees of pressure. In order to generate these instances, we implemented a configurable micro-benchmark, named access_time3, and combined four simultaneous instances of it [17]. Each instance of this micro-benchmark can perform read() and write() system calls on a large file (10 GiB) within a closed-loop according to the following parameters: *block size* (*bs*), i.e., the size of each read and write operation; *random ratio* (*rr*), i.e., the proportion of random accesses in the workload mix ($[0-1]$); and *write ratio* (*wr*), i.e., the proportion of write operations in the workload mix ($[0-1]$). To avoid the influence of the operating system's cache, each access_time3 instance only performed direct I/O operations (i.e., setting the flags O_DIRECT and O_DSYNC in the open() syscall).

We produced one set W for each *bs* evaluated in this paper: 4, 8, 16, 32, 64, 128, 256, and 512 KiB. For example, we denote $W^{bs=512}$ as the set of concurrent workloads performing I/O operations of 512 KiB. Each $W^{bs=X}$ contains 25 concurrent workloads. The first four instances (w_1 to w_4) represent a progressive increase of read-only operations: w_1 corresponds to only one instance of access_time3 configured with $wr = 0$; w_2 has two active access_time3 instances with $wr = 0$, and so on.

The remainder instances of W (w_5 to w_{25}) represent a progressive increase of write operations. For these concurrent workloads, we progressively set the parameter *wr* of each instance of access_time3 to 0.1 (from w_5 to w_8), then to 0.2 (from w_9 to w_{12}), and so on for $wr = 0.3, 0.5, 0.7$, and 1.0, following the same round-robing pattern. Once SSDs are less sensitive to random accesses than HDDs, this paper considers $rr = 0.5$ in all the evaluated experiments. The analysis of other *rr* values is left for future work.

3 EVALUATION

Using the pressure scale methodology described in the previous section, we analyze the performance interference experienced by RocksDB with different benchmarks and workloads when competing for persistent storage resources. The performance evaluation presented in this paper focuses on a single-node key-value storage engine accessing a representative high-performance SSD. The evaluation of multiple nodes and different SSD models are left for future work. The experimental setup is described in Section 3.1. Section 3.2 brings the results using one specific block size (i.e., $W^{bs=512}$), whereas the comparison between different block sizes is discussed in Section 3.3.

3.1 Experimental Setup

The experiments were conducted on a single machine with a 6-core, 12-thread AMD Ryzen 3600 CPU, 32 GB of memory, a 500 GB Samsung 970 EVO SSD NVMe, and a 500 GB 7500 RPM SATA disk. The operating system used was a GNU/Linux Ubuntu 20.04 with kernel 5.4. The NVMe device was formatted as an ext4 file system and mounted with the discard parameter activated.

From the YCSB benchmark, we used the workloads A (50% get, 50% put) and B (95% get, 5% put). The number of records used in this benchmark was 50 million (≈ 50 GiB), with 20 bytes for each key and 1 KiB for data. From db_bench, we used the workload *readwhilewriting*, which consists of one thread to perform puts and nine threads to perform gets. In this case, we used a database with 500 million records (≈ 54 GiB), containing 20 bytes for key and 400 bytes for data.

3.2 Performance Interference with $W^{bs=512}$

This subsection presents the performance impact of our pressure scale using $W^{bs=512}$. For each benchmark and workload mentioned before, we use an experiment window of 60 minutes after 30 minutes of warm-up. After this warm-up period, all the analyzed benchmarks are in steady-state, which was experimentally verified by running the same benchmarks without concurrent workloads for 180 minutes. This steady-state analysis was omitted due to space constraints [17, plot/exp_db.ipynb].

To define w_0, the first 10 minutes of each experiment were reserved for running the RocksDB's workload without concurrency. After the first 10 minutes, we started the W instances to produce an approximated progressive increase of demand for the storage resources. Each instance of W ran along two minutes, i.e., w_1 initiated at minute 10, w_2 at minute 12, w_3 at minute 14, and so on.

Figures 1a, 1b, and 1c show the throughput telemetry of RocksDB measured in transactions per second (tx/s) obtained, respectively, from YCSB, workloads A and B, and db_bench when competing for the same persistent performance device with $W^{bs=512}$. We also plot in these graphs the mean value of each 2 minutes interval. Additionally, the labels at the top of each figure indicate when each $w_i \in W^{bs=512}$ starts.

The Figs. 1d, 1e, and 1f correspond to the same experiments presented by the Figs. 1a, 1b, and 1c. These graphs show the average throughput of RocksDB for each $w_i \in W^{bs=512}$ (left axis) in addition to the corresponding pressures normalized by the w_0 of each experiment (right axis). At the bottom of each figure, we also plot the same normalized pressure of each w_i sorted by $\preceq_W$ (Def. 2). Once normalized pressure corresponds to the performance degradation relative to w_0, the following discussion uses these values in terms of percentage, i.e., normalized pressure ×100.

We observe from Fig. 1 distinct performance trends for each benchmark. The read-only concurrent workloads (w_1 to w_4) exerted a marginal effect on db_bench (Fig. 1f), being significantly

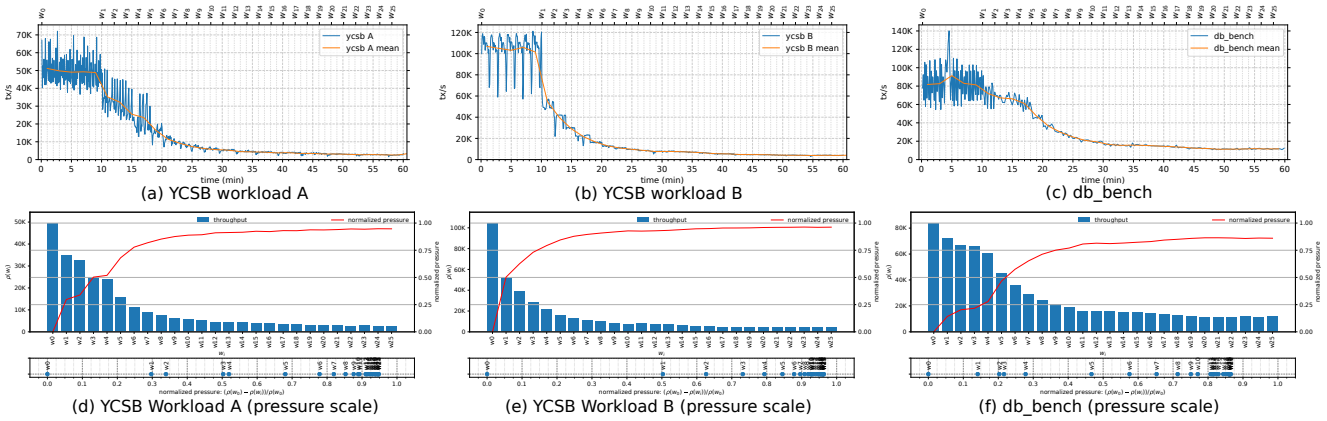

Figure 1: The performance interference on the RocksDB's benchmark and workloads ($W^{bs=512\text{KiB}}$).

more aggressive on YCSB workload A and B with a performance reduction of 30% and 50% in w_1, respectively (Figs. 1d and 1e). As the concurrent write requests gradually increased from w_5 to w_{25}, the RocksDB's performance further degraded in all benchmarks, with a moderated reduction for db_bench (47%) and a more severe degradation for YCSB workload B (84%) in w_5. These results demonstrate the high sensitivity of RocksDB when competing for the same flash storage device with other workloads.

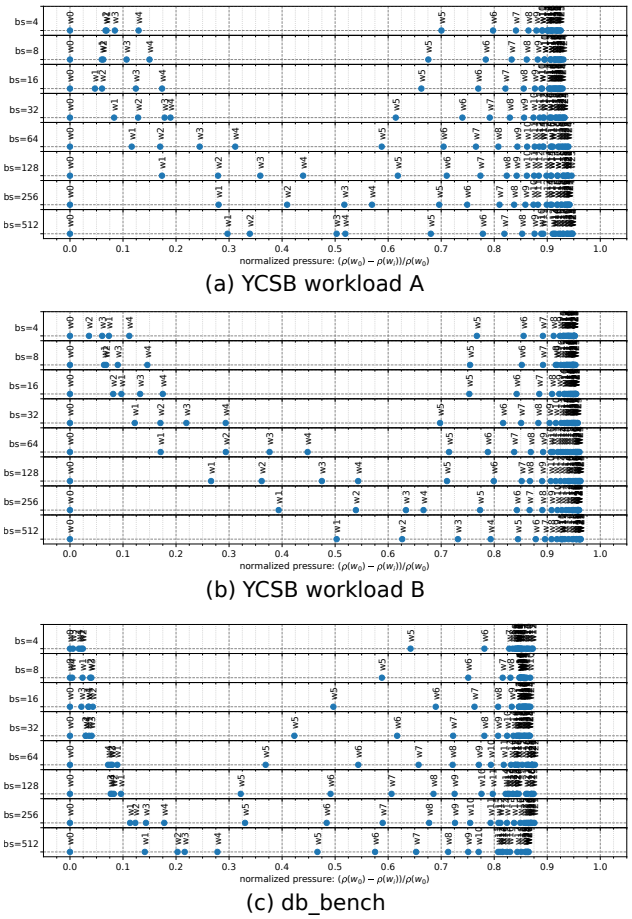

Figure 2: Pressure scale with different block sizes.

3.3 Block Size Effect on the Pressure Scale

This subsection analyzes the effects of our pressure scale with different block sizes, i.e., bs = 4, 8, 16, 32, 64, 128, 256, and 512 KiB. Due to space constraints, this paper only presents the normalized pressure scale. All the remainder graphs produced from these experiments are available in the project's repository [17, plot/exp_db.ipynb].

Figures 2a, 2b, and 2c show the normalized pressure scale of all evaluated block sizes in concurrency with YCSB workloads A and B, and db_bench, respectively. From these graphs, we observe a clear distinction between the read-only concurrent workloads (w_1 to w_4) and the workloads with write operations (w_5 to w_{25}), which we discuss in the following.

Read-only concurrent workloads. For the three benchmarks and workloads analyzed, the concurrent workloads w_1 to w_4 produced little pressure in the experiments with small block sizes. In most cases, the distance between w_1 and w_4 was also smaller if compared to larger block sizes, with some eventual order inversions, especially with respect to w_1. Due to the small performance difference produced by some concurrent workload instances, the performance fluctuation presented by the key-value store is the most probable cause of these order inversions [19].

The small pressure produced by the read-only concurrent workloads in the experiments with small block sizes reveals the intrinsic parallelism of flash devices when handling small concurrent read requests. These results suggest that RocksDB may be co-located with these workload types without significantly degrading performance. On the other hand, concurrent workloads with large read requests must be avoided.

Write concurrent workloads. We observe from Fig. 2 that the introduction of concurrent write operations produced more pressure in the experiments with small and large block sizes. For YCSB workloads A and B, the cases with the lowest pressures occurred in the experiments with bs = 32, 64, and 128 KiB, whereas the db_bench presented better results with bs = 128 and 256 KiB. Even

in the best cases, however, the introduction of concurrent write requests represented significant performance degradation for YCSB workloads A (59%) and B (70%). For db_bench, the degradation produced by w_5 was inferior to 35% in the best cases (bs = 128 and 256), but rapidly dropped performance with w_6 and w_7.

Although concurrent read-only workloads with small block sizes had little impact on RocksDB's performance, the introduction of small write requests in the concurrent workload mix may represent serious performance concerns. In these cases, the pressure difference between the read-only (w_1 to w_4) and the read-write concurrent workloads (w_5 to w_{25}) was significantly large in all evaluated benchmarks. Furthermore, the pressure produced by w_5 and w_6 in $W^{bs=4}$ was superior to w_5 and w_6 in $W^{bs=512}$ for YCSB workload A (Fig., 2a) and db_bench (Fig., 2c).

The above results suggest that RocksDB may be better co-located with read-only concurrent workloads that perform small (≤ 16KiB) read requests. The presence of concurrent write requests may represent serious performance issues depending on the workload submitted to the key-value store. In this case, the better choice is to consider concurrent write requests with larger block sizes, i.e., between 32 and 128 KiB, or 256 KiB in some cases. Smaller and larger write requests must be avoided for co-located workloads.

The performance behavior produced by our pressure scale with different block sizes demonstrates how our methodology can stress the intrinsic characteristics of the evaluated storage device. Such characteristics include out-of-place updates, the use of garbage collector, as well as the internal organization of flash memory (i.e., pages, blocks, dies, and packages) and its respective communication with the device controller (i.e., parallel channels and ways) [14].

4 RELATED WORK

Diverse studies have investigated key-value stores' performance and its relation to persistent storage devices. Luo and Carey [19] analyzed the performance stalls caused by merge-operations and proposed a scheduler based on I/O bandwidth budgets. Yoon et al. [23] considered different storage devices for hot and cold data to balance workload performance and storage costs. Other works also analyzed the performance variance of key-value stores in database transactions [15, 16], query processing [1, 2, 6], and Google's workloads [9]. Nevertheless, none of these studies consider performance interference on the storage device, which is critical in the presence of concurrent workloads.

Z. Cao et al. [5] recently performed an in-depth analysis of RocksDB workloads at Facebook, disclosing that the key-value pairs' distribution and space localities play a significant role in operational environments. Other works also evaluated the performance and implementation of realistic workloads [4, 10–12, 18]. However, these studies did not consider key-value stores disputing the persistent storage device with other tenants, critical in today's cloud computing environments.

Although there are several benchmarks designed to evaluate key-value stores' performance, including YCSB [8], LinkBench [3], BigDataBench [22], and db_bench [13], the assessment of performance interference caused by concurrent workloads is still missing in these projects. Other benchmarks and frameworks are designed to coordinate multi-tenant workloads, such as SPEC's Cloud IaaS [21] and CloudBench [20], but they do not evaluate the intrinsic characteristics of flash-based storage devices.

5 CONCLUSION

This paper has analyzed the performance interference on a key-value store deployed in a multi-tenant environment. Through a systematic methodology named "pressure scale" that generates proper concurrent workloads over time, we revealed the performance impact that a key-value storage engine may suffer when competing for persistent storage resources based on flash memory.

We conducted several experiments using the proposed concurrent workloads with two representative benchmarks, i.e., YCSB, workloads A and B, and db_bench. The results reveal how different access patterns may affect the key-value store's performance, especially with respect to the block size and the proportion of write operations in the workload mix. For the read-only concurrent workloads, the key-value store exhibited less performance degradation with small block sizes. The read/write concurrent workloads presented significant performance degradation in most of the experiments, especially in the cases with small and very large block sizes. The significant performance degradation observed in most of the experiments emphasizes the importance of measuring and managing performance interference in multi-tenant environments.

REFERENCES

[1] M. Armbrust, K. Curtis, and T. Kraska et al. 2011. PIQL: Success-Tolerant Query Processing in the Cloud. *Proc. VLDB Endow.* 5, 3 (2011), 181–192.
[2] M. Armbrust, E. Liang, and T. Kraska et al. 2013. Generalized scale independence through incremental precomputation. In *SIGMOD 2013*.
[3] T. G. Armstrong, V. Ponnekanti, D. Borthakur, and M. Callaghan. 2013. LinkBench: a database benchmark based on the Facebook social graph. In *SIGMOD 2013*.
[4] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. 2012. Workload analysis of a large-scale key-value store. In *SIGMETRICS 2012*.
[5] Z. Cao and S. Dong et al. 2020. Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook. In *USENIX FAST 2020*.
[6] S. Chaudhuri, H. Lee, and V. R. Narasayya. 2010. Variance aware optimization of parameterized queries. In *SIGMOD 2010*.
[7] B. F. Cooper. 2020. YCSB. https://github.com/brianfrankcooper/YCSB.
[8] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. 2010. Benchmarking cloud serving systems with YCSB. In *SoCC 2010*.
[9] J. Dean and L. A. Barroso. 2013. The tail at scale. *Commun. ACM* 56, 2 (2013), 74–80. https://doi.org/10.1145/2408776.2408794
[10] B. K. Debnath, S. Sengupta, and J. Li. 2010. FlashStore: High Throughput Persistent Key-Value Store. *Proc. VLDB Endow.* 3, 2 (2010), 1414–1425.
[11] B. K. Debnath, S. Sengupta, and J. Li. 2011. SkimpyStash: RAM space skimpy key-value store on flash-based storage. In *SIGMOD 2011*.
[12] G. DeCandia, D. Hastorun, and M. Jampani et al. 2007. Dynamo: amazon's highly available key-value store. In *SOSP 2007*.
[13] Facebook. 2020. RocksDB. https://rocksdb.org/.
[14] D. Gouk, M. Kwon, and J. Zhang et al. 2018. Amber*: Enabling precise full-system simulation with detailed modeling of all ssd resources. In *MICRO 2018*.
[15] J. Huang and B. Mozafari et al. 2017. A Top-Down Approach to Achieving Performance Predictability in Database Systems. In *SIGMOD 2017*.
[16] J. Huang, B. Mozafari, and T. F. Wenisch. 2017. Statistical Analysis of Latency Through Semantic Profiling. In *EuroSys 2017*.
[17] A. Lange. 2020. Rocksdb_test. https://github.com/alange0001/rocksdb_test.
[18] H. Lim, B. Fan, D. G. Andersen, and M. Kaminsky. 2011. SILT: a memory-efficient, high-performance key-value store. In *SOSP 2011*.
[19] C. Luo and M. J. Carey. 2019. On Performance Stability in LSM-based Storage Systems. *Proc. VLDB Endow.* 13, 4 (2019), 449–462.
[20] M. Silva, M. R. Hines, and D. et al. Gallo. 2013. CloudBench: Experiment Automation for Cloud Environments. In *IC2E 2013*.
[21] SPEC. 2018. Cloud IaaS 2018. https://www.spec.org/cloud_iaas2018
[22] L. Wang, J. Zhan, and C. Luo et al. 2014. BigDataBench: A big data benchmark suite from internet services. In *HPCA 2014*.
[23] H. Yoon, J. Yang, and S. F. Kristjansson et al. 2018. Mutant: Balancing Storage Cost and Latency in LSM-Tree Data Stores. In *SoCC 2018*.