When Database Meets New Storage Devices: Understanding and Exposing Performance Mismatches via Configurations

Reading (first reading)

What field or direction does this article belong to?

Database and nvms

What problem was solved? Why is this issue so important?

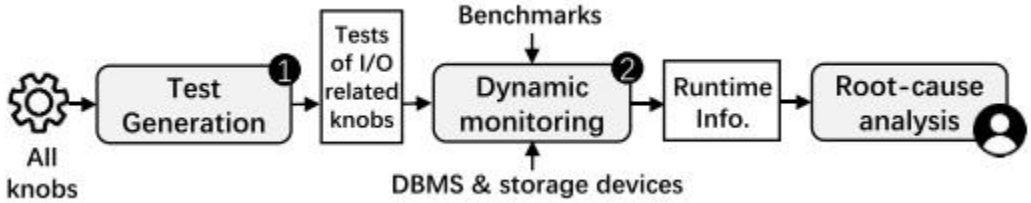
Problem: The existing database management system may not be able to perform as well as the new storage device should (**high-performance storage devices often provide lower than expected performance**), but it has not been solved; (this problem does not seem to be that important)

What methods and models were used? Why does this method solve this problem?

Experimental design: The control variable is the device (nvme, SSD, HDD), and the effect is tested on the same DBMS
testing platform:

- MySQL, PostgreSQL, SQLite, MariaDB, MongoDB and Redis
- YCSB and TPC

Test Methods:



Generate the workload, adjust only the knobs related to IO, and get the reasons by analyzing the running results.
Testing process:

- I/O related knob confirmation
  - I/O related system calls
  - Connect knobs to I/O related system calls
  - data flow connection
  - control flow connection
- Generate the value of the knob

- Build a search space
- Testing principles: **Heuristic rules: Knob mechanism (optimization) -> High-level devices perform worse than low-level devices (relative value) -> Performance mismatch (core logic chain finds performance mismatch)**
- Determine the root cause: Trace the I/O path, hypothesize a few possible factors, and then compare.

It is essentially a control variable method. The author assumed several possible parameter configurations, and then found that changing these parameter configurations (compared with each other) can find that the performance of the new storage device decreases much more than normal.

What is the core conclusion? What else can be done next?

In this article, we discover that shoehorning new storage devices into an existing DBMS can result in a performance mismatch that can have a serious impact on performance. Performance mismatches are rarely studied and/or detected. To fill this gap, we conducted a comprehensive study of performance mismatch to understand its symptoms, root causes, and triggers. In this study, we propose a method that leverages configurations to detect performance mismatches. We found that performance mismatches can be divided into three types based on their root causes and we conducted an in-depth analysis of the root cause patterns. Compared with baseline methods, our method is more efficient and can detect more performance mismatches.

Experimental results

Test Information					Results					
DBMS	**Version**	**# Knobs**	**Workload**	**# Tests**	**Cost**	**# Violation**	**# PPM**	**# Public**	**# Knobs touch**	**Penalty**
MySQL	8.0.22	529	R	21,233	124d	69	13 (2)	2	8	14%-94%
			W	2,915	42d	52	19 (0)	4	12	12%-146%
PostgreSQL	12.11	256	R	8,965	55d	55	12 (4)	1	5	78%-215%
			W	1,230	19d	41	13 (3)	0	8	16%-122%
SQLite	3.36.0	74	R	2,836	21d	-	0 (0)	0	-	-
			W	389	7d	14	4 (1)	0	3	21%-33%
MariaDB	10.6.3	418	R	16,851	90d	55	11 (2)	1	8	12%-42%
			W	2,313	31d	44	13 (0)	1	12	16%-192%
MongoDB	5.0.0	169	R	4,045	51d	31	8 (2)	0	2	-
			W	1,906	26d	24	8 (2)	1	2	43%-67%
Redis	6.0	97	R	2,305	29d	29	10 (2)	0	2	-
			W	1,104	15d	18	12 (2)	0	3	22%-84%
Total	-	1,543	-	66,092	510d	432	123 (20)	10	56†	12%-215%

 # Knobs: number of all knobs; **Workload**: "W" write-intensive (TPC-C and YCSB-A & F), "R" read-intensive (TPC-E/H/DS and YCSB-B~E); **# Tests**: number of tests conducted; **Cost**: time consumption in machine days; **# Violation**: number of tests that violate the heuristic rule in § 2.2; **# PPM**: number of potential performance mismatches exposed (may duplicate), $_{(\cdot)}$: PPMs found by cross-check (i.e., *false negatives* of the rule); **# Public**: number of PPMs can be found in public areas; **# Knobs touch**: number of knobs touched by those PPMs; **# Penalty**: performance penalty in new devices of those PPMs; †de-duplicated sum.

exp: If after changing the knob, the query latency drops to 1.3x in the HDD and to 2.6x in the NVMe SSD, the performance penalty for this mismatch is 100% · 2.6/1.3 = 200%

Three major reasons

Size in write

The forced write request issued by the DBMS is too small, causing the SSD's write cache to work inefficiently.

Parallelism in writing and reading

I/O requests issued by the DBMS are almost always serialized, causing the internal parallelism of

the new device to be largely wasted.

Reading and writing order

DBMS usually consumes a lot of resources (CPU, memory) to convert random I/O into sequential I/O; while in NVMe SSD, the speed difference between the two types of I/O is small, which makes the conversion efficiency less Low or even wasteful.

doubt

How to detect performance mismatch, what is the standard, and is this standard reliable?

There is no standard, the author defines a heuristic rule (similar to finding differences).

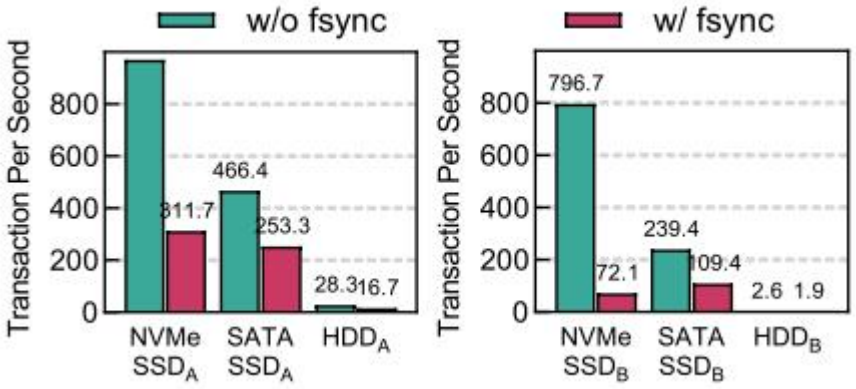
What is the impact of performance mismatch, and is this impact important?

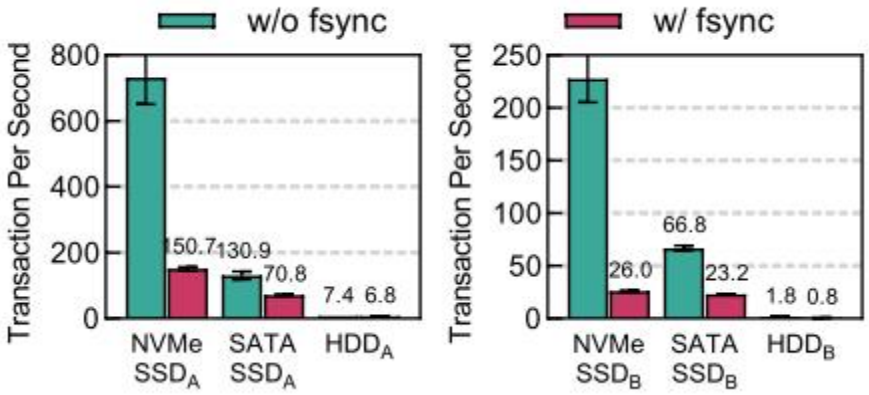
It may cause the performance of equipment such as nvmes to not perform as expected, but it is still better than traditional equipment, so there does not seem to be a bad impact (excessive performance)

Is the detection method reasonable and comprehensive?

Personally, I think the experimental design is more reasonable and all factors considered.

Definition of performance mismatch





The base numbers of nvmes and sata are inherently larger than HDD. Is this difference amplified due to hardware reasons rather than configuration reasons?

Summarize

The existing DBMS was unable to bring out the best performance of the new storage device, and three root causes were found by changing the DBMS parameter configuration.

Personally, I think the highlight of this article is the experimental design. Although it only considers the parameter configuration of the DBMS, the control experiment is complete and persuasive. Secondly, the workload is relatively large. However, even if the existing DBMS cannot bring out the best performance of the new storage device, it still has a huge improvement in throughput and other performance compared to traditional devices such as HDD (just slightly worse than itself). DBMS developers are really motivated to improve these Configure? Personally, I prefer the latter between adapting existing DBMS to new equipment and developing new DBMS to adapt to **new** equipment **.**