ACCELERATING CRUD WITH CHRONO DILATION FOR TIME-SERIES STORAGE SYSTEMS

Lan Nguyen, **Ioan Raicu** (Advisor)

Illinois Institute of Technology {lnguyen18@hawk.iit.edu, iraicu@iit.edu}

ILLINOIS TECH
College of Computing

DataSys
Data-Intensive Distributed Systems Laboratory

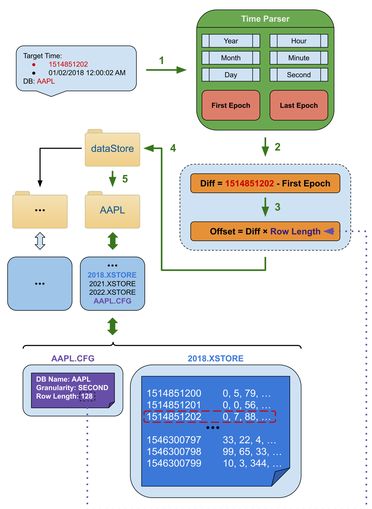
ABSTRACT

The increase in high-precision, high-sample-rate time-series data (e.g. health data, financial price data) is a challenge to existing database technologies. We have developed a novel technique that utilizes sparse-file support to achieve O(1) time complexity in create, read, update, and delete (CRUD) operations while supporting time granularity down to 1-second. We designed and implemented XStore to be lightweight and offer high performance without the need to maintain an index of the time-series data. We have conducted an evaluation between XStore and MongoDB using synthetic data spanning 20 years, with second granularity, totaling over 5 billion data points and XStore achieves 2.5X better latency and delivers up to 3X improvement in throughput.

MOTIVATIONS

- Chrono structure – Accelerate performance through the adoption of a heavily structured time-series data
- Tree structure – The elimination of tree structure usage provides superior savings in system's resources
- Scalability – Achieve constant time complexity irrespective of granularity/size
- Index – Sustain high performance across workloads without index

METHODOLOGY



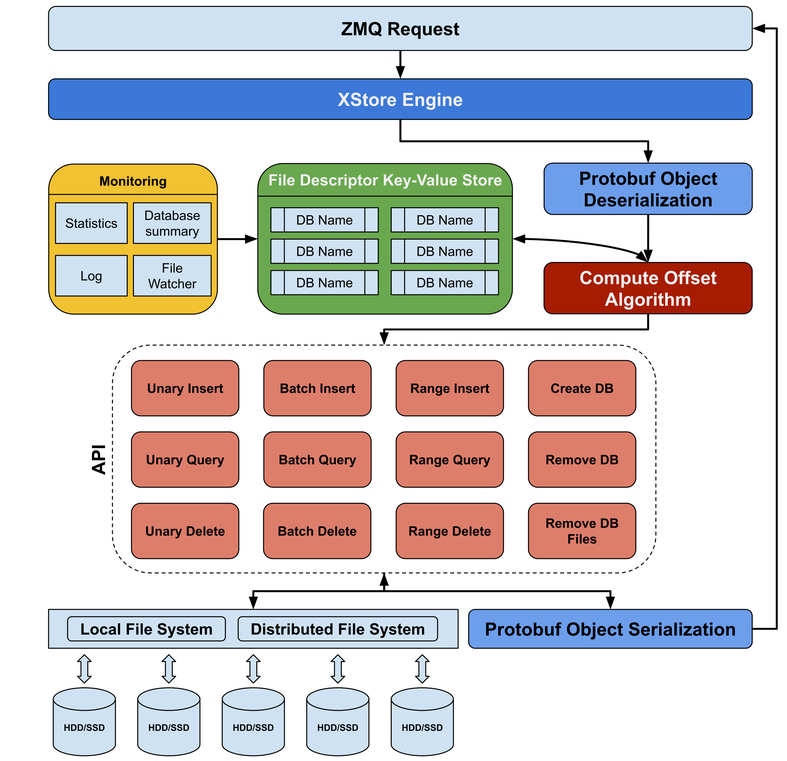
TESTBED [1]

- 2x Intel® Xeon® E5-2670 v3 Processor – @2.30GHz (12 cores, 24 threads)
- 8x 16GB (128 GB) of DDR4-2,133 ECC Registered RAM
- 1x Seagate ST9250610NS SATA 7,200 RPM HDD
- Broadcom NetXtreme II BCM57800 1/10 Gigabit Ethernet
- Linux Ubuntu 22.04 LTS
- Filesystem: EXT4

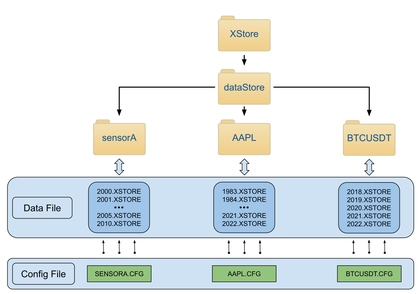
DESIGN & IMPLEMENTATION

The core idea is to utilize sparse file support along with orders of magnitude of time to achieve O(1) time complexity in create, read, update, and delete (CRUD) operations while supporting time granularity down to nanosecond.

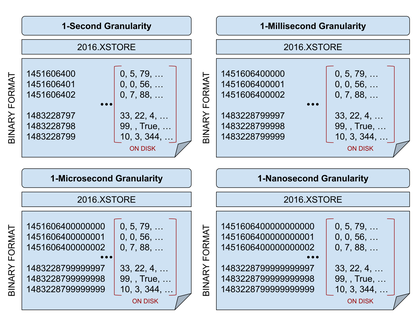
The constant time complexity is stemming from the fact that an exact location of a given timestamp can be located using a simple arithmetic operation.



File Organization

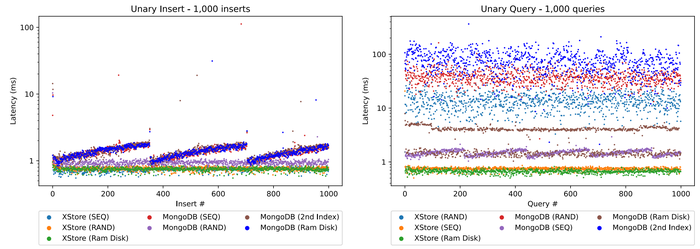


Data Organization

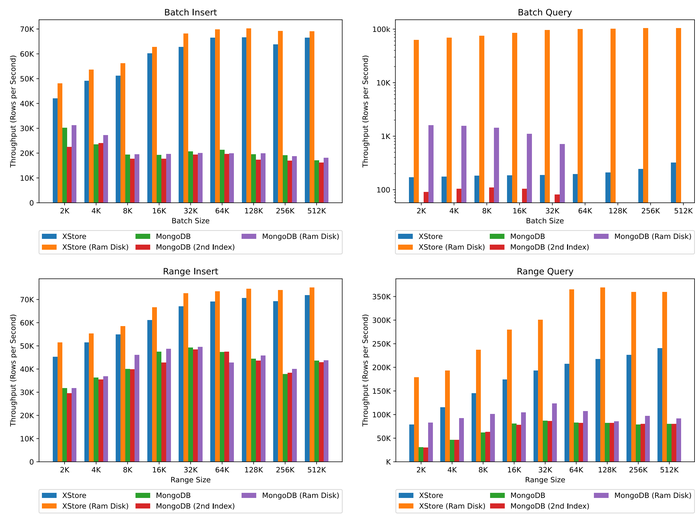


EXPERIMENTS

LATENCY



THROUGHPUT



CONCLUSION

According to the empirical experiments presented, we have seen notable improvements in performance across all workloads. Upholding the claim of our technique in expediting CRUD operations using sparse file coupled with time dilation.

On the contrary, we deem it is critical to further investigate the performance impact of sparse file. There are multiple exciting avenues that we could explore as our next steps as following:

- Incorporate compression and archive mode
- Enable generalized search with XSearch/SCANNS [2]

REFERENCES

[1] Kate Keahey et al. "Lessons Learned from the Chameleon Testbed". In: *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association, July 2020.

[2] Alexandru Iulian Orhean et al. "SCANNS: Towards Scalable and Concurrent Data Indexing and Searching in High-End Computing System". In: *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE. 2022, pp. 51–60.

[3] Wikipedia contributors. *Sparse file — Wikipedia, The Free Encyclopedia*. [Online; accessed 3-August-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Sparse_file&oldid=1163967763.