Accelerating CRUD with Chrono Dilation for Time-Series Storage Systems

Lan Nguyen
Illinois Institute of Technology
Chicago, IL, USA
lnguyen18@hawk.iit.edu

Ioan Raicu (Advisor)
Illinois Institute of Technology
Chicago, IL, USA
iraicu@iit.edu

Abstract

In recent years, we have seen an un-precedented growth of data in our daily lives ranging from health data from an Apple Watch, financial stock price data, volatile crypto-currency data, to diagnostic data of nuclear/rocket simulations. The increase in high-precision, high-sample-rate time-series 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[1] to be lightweight and offer high performance without the need to maintain an index of the time-series data. We have conducted a detailed evaluation between XStore and existing best-of-breed systems such as MongoDB using synthetic data spanning 20 years, with second granularity, totaling over 5 billion data points. Through empirical experiments against MongoDB, XStore achieves 2.5X better latency and delivers up to 3X improvement in throughput.

CCS Concepts: • **Information systems** → **Key-value stores**; **Point lookups**; **Record and block layout**.

Keywords: data storage, sparse file, time-series, key-value store, file systems, simulation, CRUD

ACM Reference Format:
Lan Nguyen and Ioan Raicu (Advisor). 2023. Accelerating CRUD with Chrono Dilation for Time-Series Storage Systems. In *Proceedings of The International Conference for High-Performance Computing, Networking, Storage, and Analysis (SC23).* ACM, New York, NY, USA, 3 pages.

[1] https://gitlab.com/lvn2007/XStore

SC23, November 12–17, 2023, Denver, CO

1 Introduction

In recent years, we have seen an un-precedented growth of data in our daily lives ranging from health data originates from smart watch to financial stock price data. Granted that there exists a diverse set of database/storage system solutions designed to support time-series data. It has come to our attention that solutions such as MongoDB employs B-Tree or similar for its storage engine. Incurring a logarithmic cost of traversing a tree but also cost of searching linearly through each document to locate a correct target within a time bucket [1, 2].

With that said, we are motivated to identified and addressed the following but not limited to:

- Chrono structure – Accelerate performance through the adoption of a heavily structure 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

2 XStore System Architecture

We have developed a technique that utilizes sparse-file support to achieve constant time complexity in create, read, update, and delete (CRUD) operations while making it agnostic to time granularity. We designed and implemented XStore to be lightweight and offer high performance without the need to maintain an index of the time-series data. XStore is a time-series specialized key-value store database, a type of a *NoSQL* database program designed to address an ever-growing amount of time-series data.

Storing and retrieving data is a 2-step process for any particular row of data given a target timestamp. By computing the file path based on the target timestamp, we can determine the exact location of a target data file resize on a physical storage medium. Once a target file has been located, we proceed with computing an offset position of a given timestamp. Upon completion, we have acquired a definitive location of data that is associated with a specific timestamp. This approach allows us to implement the time-series storage system without the need for an explicit index to be created

or maintained. For visual representation of the data organization in XStore, see Figure 1. In summary XStore posses the following attributes:

- Relies on sparse file – Enable the ability to only physically occupy space on disk with actual data
- Evaluated with 128 bytes per row (fully configurable)
- Support for unlimited column in a row
- Data is stored in binary file format without compression for optimized space efficiency and performance

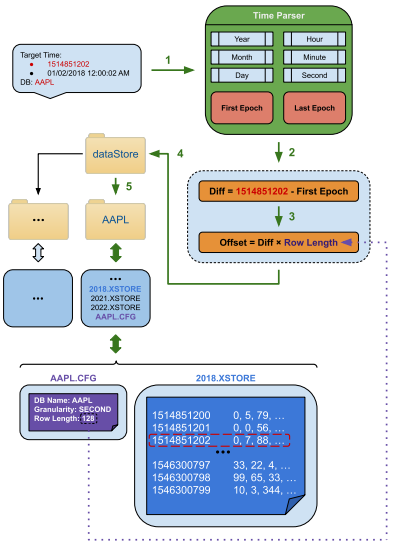

Figure 1. XStore Organization

3 Performance Evaluation

To validate our proposed methodology, we ran a handful of benchmarks with a focus in latency and throughput. The benchmarks were developed and ran on four dedicated Chameleon Cloud [3] and Mystic [5] baremetal instances as client (in Python) and server for both XStore and MongoDB respectively. All four instances have identical specifications, that is:

- 2x Intel® Xeon® E5-2670 v3 Processor – @2.30GHz
- 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

Throughout all tests, we employed synthetic data (≈ 175GB) ranges over 20 years from 2000–2020 with ≥ 663M rows and 8 columns. Each row is a key-value pair and ≈ 264 bytes long. A key is represented as a timestamp (8 bytes) and values totaling ≈ 256 bytes. Each column is a hash value (32 bytes) produced by concatenates the value of timestamp with column index i.e., [946684800-1, 946684800-2, ..., 946684800-8].

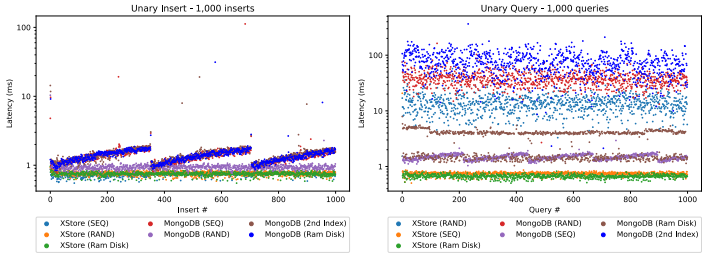

Figure 2. Latency workloads

In Figure 2, we observe an improvement of 1.5-2X comparing to MongoDB. Although, both XStore and MongoDB employs write-delaying strategy. The cost of MongoDB's time bucket strategy was highlighted in both read/write workload due to incurred traversal costs. In random query, XStore consistently delivers a latency of 13.89 (ms); signifies the low overhead cost pertaining to our methodology since disk's average read/write latency is ≈ 9.5 (ms) [4].

In Figure 3, XStore outperforms MongoDB by 0.5-4X across all workloads and performance is saturated once batch/range size is above 32K. Nevertheless, XStore's capability can further extend beyond 512K size as shown in range query workload. On the other hand, across all workloads, we notice a pattern such that a gap in performance between XStore and MongoDB as size increases.

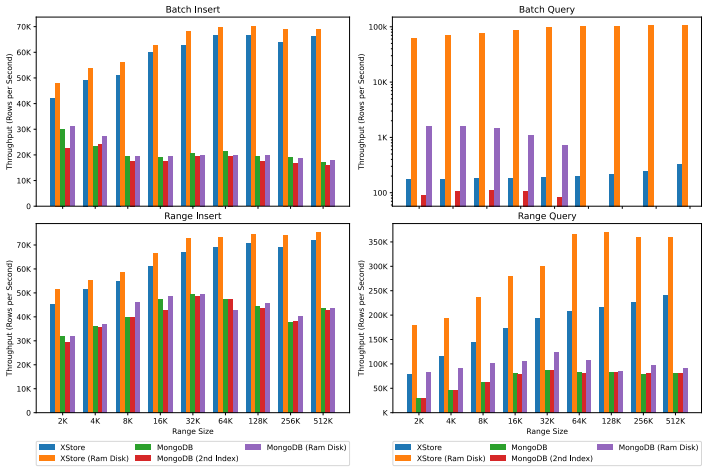

Figure 3. Throughput workloads

4 Conclusion and Future Work

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: record level compression and generalized search.

References

[1] Carlos Garcia Calatrava, Yolanda Becerra Fontal, Fernando M Cucchietti, and Carla Diví Cuesta. 2021. NagareDB: A resource-efficient document-oriented time-series database. *Data* 6, 8 (2021), 91.

[2] MongoDB Inc. [n. d.]. *Set Granularity for Time Series Data – MongoDB Manual.* https://www.mongodb.com/docs/rapid/core/timeseries/timeseries-granularity/

[3] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. 2020. Lessons Learned from the Chameleon Testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association.

[4] Seagate Technology LLC. [n. d.]. *Constellation.2 Data Sheet.* https://www.seagate.com/www-content/product-content/constellation-fam/constellation/constellation-2/en-gb/docs/constellation2-fips-data-sheet-ds1719-4-1207gb.pdf

[5] AI Orhean, A Ballmer, T Koehring, K Hale, XH Sun, O Trigalo, N Hardavellas, S Kapoor, and I Raicu. 2019. Mystic: Programmable systems research testbed to explore a stack-wide adaptive system fabric. In *8th Greater Chicago Area Systems Research Workshop (GCASR)*.