# KungfuDB: a low latency in-memory time series database for quantitative trading systems

Xiaodong Li[1(✉)], Yan Zhou[1], Wenkai Liu[2], Yizhi Zhang[2], and Keren Dong[2]

[1] College of Computer Science and Software Engineering,
Hohai University, Nanjing, China
{xiaodong.li, y.zhou}@hhu.edu.cn
[2] Kungfu Origin Technology, Beijing, China
{wenkai.liu, yizhi.zhang, keren.dong}@kungfu.link

**Abstract.** Modern quantitative trading systems rely on low-latency time series databases for critical tasks such as tick data writing, reading and persistence, which support facilities like trading signal analysis and strategy back-testing. However, existing databases, even top-ranked solutions on DB-Engine, lack specialized optimizations required by these financial applications. To address this need, Kungfu Origin Technology and Hohai University develop KungfuDB, a low latency in-memory financial time series database. KungfuDB uses innovative log and page management algorithms and zero-copy memory techniques to achieve ultra-fast database operations with minimal latency. Experiments against leading databases like InfluxDB and TimescaleDB show that KungfuDB delivers up to 60x performance improvements in writing and reading tick data. Notably, KungfuDB is open-sourced in Github and successfully deployed by leading securities and fund management firms in China.

**Keywords:** Time series database · Quantitative trading · Low latency

## 1 Introduction

Modern quantitative trading systems relentlessly strive for ever-increasing speed. The most time-consuming operations in these systems are writing, reading, and persistence of various data, such as market data, transaction records, and system logs. This drives an urgent demand for low-latency time series databases. However, existing time series databases fall short in addressing these demands. Time series databases, such as InfluxDB[1] and TimescaleDB[2], incorporate multi-objective optimization in their storage processes, but lack clear task prioritization and specialized optimizations for database operations in quantitative trading systems. These limitations hinder their efficiency in financial applications, underscoring the need to design a low-latency financial time series database.

The design of a low-latency financial time series database must address two key issues. First, which tasks should it prioritize? IoT time series databases [4,5]

[1] InfluxDB: https://www.influxdata.com/
[2] TimescaleDB: https://www.timescaledb.cn/

prioritize edge computation and data storage, significantly reducing network transmission and improving efficiency. Systems like Facebook's Gorilla [3] emphasize aggregation operations, leveraging cold-hot data management and multi-level data compression algorithm to speed up analysis and reduce storage usage. General purpose time series databases support a variety of tasks like CRUD operations, but lack of task prioritization makes them unsuitable for low-latency write and read requirements of quantitative trading systems. Second, what techniques should be used for data persistence? As noted in [4], industrial applications often retain all historical data for auditing and analysis. Similarly, quantitative trading systems require complete data retention for post-trade analysis, imposing strict efficiency demands on data persistence. Many native and non-native time series databases use Write-Ahead Log (WAL) for periodic memory-disk synchronization. However, WAL relies on fsync, which is not designed for high throughput. Thus, the efficiency of persistence is significantly reduced.

To meet the needs of quantitative trading systems, Kungfu Origin Technology[1] and Hohai University jointly develop a low-latency in-memory financial time series database named KungfuDB. It prioritizes tick data writing and reading, and employs an innovative memory management mechanism using Journals and Pages, leveraging mmap based zero-copy technology for ultra-low-latency data operations. This approach enables near-zero latency in simultaneous memory and disk operations, supported by asynchronous mmap kernel processes. Experiments demonstrate that KungfuDB significantly outperforms databases such as InfluxDB and TimescaleDB, achieving up to 60x improvements in tick data write and read latency. KungfuDB is open-sourced in GitHub[2] and integrated into Kungfu Trader suite. Notably, it has already been adopted by leading financial institutions[3] including Huatai Securities and China Galaxy Securities.

## 2 KungfuDB

The architecture of KungfuDB is shown in Fig. 1, which includes two main modules: Page Engine for page management and Journal file structure for memory-disk interaction. It also offers user-friendly APIs for data writing and reading.

In the Page Engine, a Locator assigns each user a dedicated writing unit (Journal), accessed by a single writing thread to ensure process and thread safety. The Page Manager prepares a Page and writes metadata like Page ID into a Page Header for querying, while the Frame Manager creates a Frame with a Frame Header (storing metadata such as timestamps and data types) and a Frame Body (containing the data). Pages are fixed-length files, but to provide unlimited-length files and reduce write latency, the Page Engine uses two modules: the Precreator, which pre-creates backup Pages when usage reaches a threshold $h_1$, and the Preloader, which loads them into memory at a higher threshold $h_2$ ($h_2 > h_1$). This design reduces the write latency from microseconds to nanoseconds.

[1] Kungfu Origin Technology: `https://www.kungfu-trader.com/`

[2] GitHub: `https://github.com/kungfu-origin/kungfu`

[3] Core partners: `https://www.kungfu-trader.com/index.php/solution/`

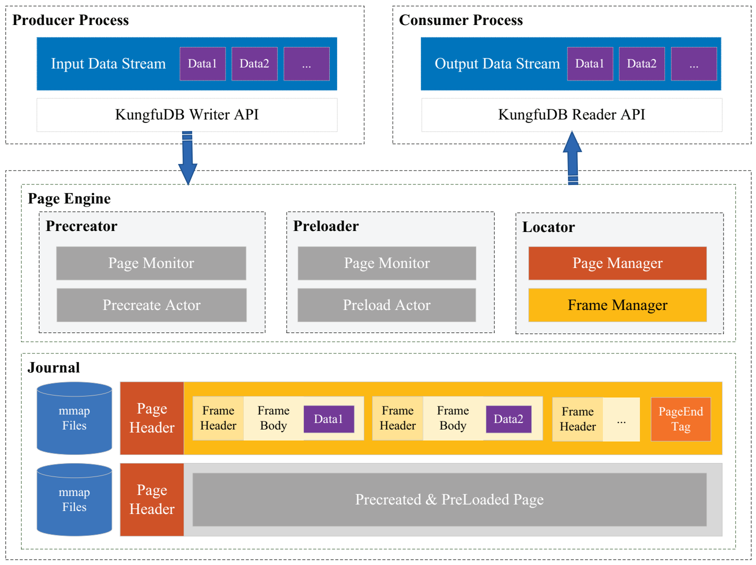

**Fig. 1.** The architecture of KungfuDB.

The writing unit Journal implements mmap files, offering a unique memory-disk synchronization mechanism. Traditional in-memory time series databases use synchronization functions like fsync during data persistence, which are time-consuming and often cause process blocking due to the unpredictability of invocation timing, resulting in higher system latency. In contrast, mmap leverages kernel-level zero-copy mmap IO. During continuous data writes, mmap retains data in memory to ensure minimal latency, while its kernel-backed IO synchronization asynchronously persists data to disk in the background. This approach achieves maximum throughput efficiency while maintaining low latency.

## 3 Demonstration and Experiments

We plan to demonstrate the proposed KungfuDB in quantitative trading scenarios, focusing on storing real-time tick data and querying historical data. Our demonstration video is available on Bilibili[1] and YouTube[2]. This demo highlights KungfuDB's ability to efficiently handle real-time tick data ingestion and asynchronous persistence, as well as its capability to quickly retrieve and extract historical data. With its low-latency performance, KungfuDB is well suited for tasks such as market data playback and strategy back-testing, significantly reducing the read and write overhead of quantitative trading systems.

[1] Bilibili: `https://www.bilibili.com/video/BV15gzyYpEk7/?vd_source=49053dfe0d51e424494378476e1d02a9`

[2] YouTube: `https://www.youtube.com/watch?v=ocrnlfg6QFo&ab_channel=YizhiZhang`

We also conduct experiments to compare KungfuDB with InfluxDB and TimescaleDB. The experiments are carried out on an HP ProLiant DL360 server. Existing benchmarks for time series databases [1,2] primarily focus on DevOps and IoT scenarios, lacking specialized designs for quantitative trading. To address this gap, we specifically design tasks for writing and reading order and quote data streams. These tests evaluate data write/read latency under varying ticker counts and data volumes, with latency as the evaluation metric.

**Table 1.** Results for data write/read with varying ticker counts and data volumes per ticker. "X ticker, Yk" means that there are X tickers and Y thousands records.

| Database | Operation | (s) | 1 ticker, 10k | 1 ticker, 100k | 10 ticker, 10k | 10 ticker, 100k | 100 ticker, 10k |
|---|---|---|---|---|---|---|---|
| **KungfuDB** | write | ave. | **0.03** | **0.33** | **0.34** | **3.94** | **3.84** |
| | | std. | 0.00 | 0.06 | 0.06 | 0.50 | 0.45 |
| | read | ave. | **0.01** | **0.06** | **0.06** | **0.60** | **0.59** |
| | | std. | 0.02 | 0.16 | 0.17 | 1.48 | 1.40 |
| InfluxDB | write | ave. | 1.22 | 7.84 | 7.98 | 80.15 | 81.18 |
| | | std. | 0.02 | 0.11 | 0.09 | 3.05 | 0.40 |
| | read | ave. | 1.74 | 20.22 | 17.08 | 237.65 | 232.63 |
| | | std. | 0.09 | 0.10 | 0.66 | 4.02 | 6.75 |
| TimescaleDB | write | ave. | 1.72 | 17.29 | 16.71 | 224.31 | 181.57 |
| | | std. | 0.07 | 0.35 | 0.13 | 26.40 | 4.13 |
| | read | ave. | 0.10 | 0.99 | 0.85 | 12.16 | 9.47 |
| | | std. | 0.01 | 0.07 | 0.03 | 5.09 | 1.13 |

The experimental results in Table 1 highlight KungfuDB's superior performance in both data writing and reading tasks. For data writing, KungfuDB outperforms both InfluxDB and TimescaleDB by 20x-60x. For data reading, KungfuDB outperforms InfluxDB and TimescaleDB by approximately 3x-60x.

**Acknowledgements.** This work was supported by the National Key R&D Program of China (grant number 2024YFC3210801).

## References

1. Hao, Y., Qin, X., Chen, Y., Li, Y., Sun, X., Tao, Y., Zhang, X., Du, X.: Ts-benchmark: A benchmark for time series databases. In: ICDE. pp. 588–599. IEEE (2021)
2. Khelifati, A., Khayati, M., Dignös, A., Difallah, D., Cudré-Mauroux, P.: Tsm-bench: Benchmarking time series database systems for monitoring applications. PVLDB **16**(11), 3363–3376 (2023)
3. Pelkonen, T., Franklin, S., Teller, J., Cavallaro, P., Huang, Q., Meza, J., Veeraraghavan, K.: Gorilla: A fast, scalable, in-memory time series database. PVLDB **8**(12), 1816–1827 (2015)
4. Wang, C., Huang, X., Qiao, J., Jiang, T., Rui, L., Zhang, J., Kang, R., Feinauer, J., McGrail, K.A., Wang, P., et al.: Apache iotdb: Time-series database for internet of things. PVLDB **13**(12), 2901–2904 (2020)
5. Yan, Y., Zheng, B., Wang, H., Zhang, J., Wang, Y.: Cnosdb: A flexible distributed time-series database for large-scale data. In: DASFAA. pp. 696–700. Springer (2023)