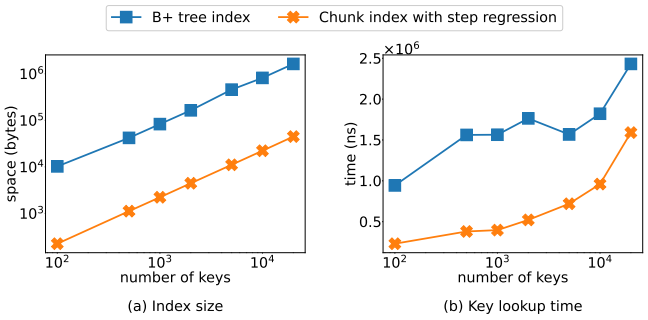


Figure 17: Comparison with B+ tree index

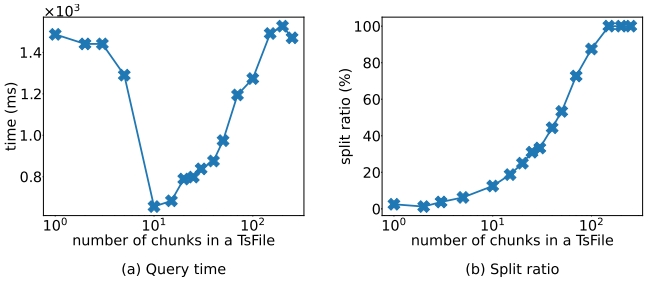


Figure 18: M4 query time with varying TsFile size

B ADDITIONAL MATERIAL

B.1 Comparison with Baseline Index

We compare the performance of our proposed chunk index with the traditional B+ tree index in terms of disk space consumption and key lookup time. The experimental results are reported in Figure 17. On average, our chunk index uses only 3% of the space of the B+ tree index to achieve twice the query speed. The high space overhead of the B+ tree index is due to its large internal nodes, while the chunk index with step regression only needs to store the first timestamp of each segment. The better key lookup performance of the chunk index also benefits from its significantly lower space overhead.

B.2 Meta-Info at the TsFile Level

The meta-info can also be maintained beyond the chunk level, e.g., in TsFile of multiple chunks. The experimental results are reported in Figure 18. When the number of chunks in a TsFile is small, e.g., only one chunk in each TsFile, it is equivalent to the case of using chunk meta-info only. The query time cost is thus high in Figure 18(a). On the other hand, with the increase of the number of chunks in a TsFile, as illustrated in Figure 18(b), the ratio of splitting a TsFile by the M4 time intervals increases, i.e., the meta-info of TsFile cannot be utilized. Thereby, for a large number of chunks in a TsFile, the corresponding time cost is high as well in Figure 18(a). In this sense, a moderate size of TsFile can improve the query performance by its meta-info across multiple chunks.

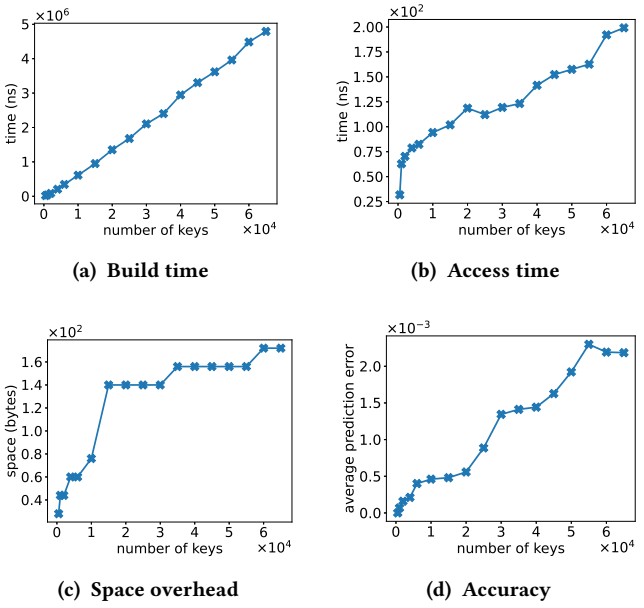


Figure 19: Evaluation of chunk index with step regression

B.3 Evaluation of the Proposed Chunk Index

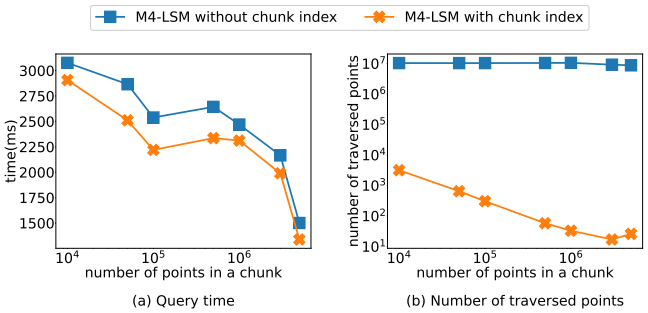
We evaluate the performance of our proposed chunk index in terms of build time, access time, space overhead, and accuracy. The experimental results on BallSpeed dataset are shown in Figure 19.

B.3.1 Build Time. As shown in Figure 19(a), the build time increases linearly with the number of keys (i.e., timestamps). This is attributed to the heuristic learning algorithm of the step regression function, which only need to traverse the data twice. The first time is to compute statistics, and the second time is to learn the step regression parameters based on statistics.

B.3.2 Access Time. As shown in Figure 19(b), the access time increases sublinearly with the number of keys. Access time is mostly the time to search for the segment where the lookup key resides. Therefore, the access time is highly correlated with the number of segments, or equally, the space overhead as shown in Figure 19(c).

B.3.3 Space Overhead. As shown in Figure 19(c), the index space grows in steps as the number of keys increases. Since the timestamps are irregular at the beginning of the datasets, there are more segments created and thus the space cost increases fast. Then, it becomes more regular with less segments and stable space cost.

B.3.4 Accuracy. As shown in Figure 19(d), the average prediction error (i.e., the average distance between the predicted and real positions of all keys) is small. The errors much smaller than 1 denote that most predicted positions are accurate. Indeed, the maximum prediction error does not exceed 5. This confirms the effectiveness of our approach for modeling real timestamp distributions. When the number of segments does not increase with the number of keys, i.e., stable space cost in Figure 19(c), the corresponding error increases in Figure 19(d). The reason is that more keys share the same segment leading to higher accumulated error.



(a) Query time (b) Number of traversed points

Figure 20: Ablation study of chunk index

B.4 Ablation Study

Ablation study of the proposed M4-LSM algorithm with and without the chunk indexing method is reported in Figure 20. It is not surprising that the number of traversed points is significantly reduced by the chunk index, as illustrated in Figure 20(b). The corresponding time cost is thus clearly improved as well in Figure 20(a). Regarding the overall downward trend of both time costs, it is because more points in a chunk mean fewer chunks under the same dataset, and hence the expected cost of candidate verification for *BP/TP* tends to decrease.

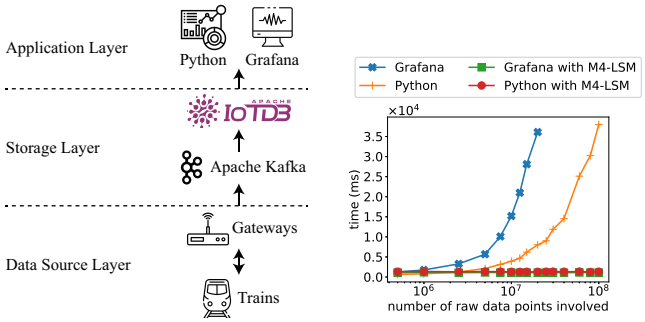
B.5 Case Study of Real Applications

We present a case study on real users practically using the systems. The customers are in various industries, including steel, manufacturing, cloud service, rail transit, and so on.

CISDI. In the steel manufacturing process of CISDI Co., Ltd., Apache IoTDB manages the time series of 250,000 sensors in 30,000 devices. The data collection interval ranges from 10 milliseconds to 5 seconds, generating 1.5 billion points every day. A dashboard is developed based on our proposed solution, to visualize the temperatures at different stages of the steel manufacturing process, including blast furnace molten iron temperature, molten steel temperature, refining entry temperature, refining exit temperature, tunnel temperature, and so on. Domain experts inspect the visualized time series of temperatures to explore the potential gaps among different stages. By optimizing the steel manufacturing process, such temperature gaps are expected to be eliminated to improve energy efficiency.

Taiji. Apache IoTDB is employed in Taiji Co., Ltd. to store the high frequency device test data at nanosecond level. Using the IEEE 1588 Precision Time Protocol (PTP), the data collection frequency is as high as 20kHz to 400kHz, generating more than 1T time series in each test task. Our proposed solution is used to visualize and compare the overall performance metrics with those of each part. The domain experts can thus evaluate whether the changed design improves the overall/part performances.

Yonyou. Apache IoTDB is used as the time series database in the Application Performance Monitoring (APM) product called YouYunYin [https://yyy.yonyoucloud.com] by Yonyou Network Technology Co., Ltd. It monitors multiple metrics of the hosts in every 20 milliseconds to 10 seconds, including CPU and network utilization,



(a) System deployment (b) Visualization performance

Figure 21: Case study of CRRC

GVM and heap memory usage, etc. The time series collected by a single metric in one day can reach 4.32 million points. Our solution is employed to visualize multiple metrics on the same dashboard for fault diagnosis of application performance. For example, by comparing the garbage collection time and the heap usage trend, users can analyze memory leaks.

CRRC. In the intelligent operation and maintenance system of urban rail vehicles in CRRC Corp., Ltd., Apache IoTDB is used to manage the sensor data collected in trains. Each train has tens of thousands of sensors installed, measuring current, voltage, pressure, speed, acceleration, temperature, and so on. The data are usually collected in every hundreds of milliseconds, generating about 48TB time series each year. Figure 21a presents an overview of system deployment in the company.

In fault diagnosis, our proposed solution is used to visualize vibration signals at a frequency about 100Hz in Grafana. Domain experts compare the suspicious data with historical data, in different granularity (e.g., hour, day, week and month), to analyze the abnormal behaviors. As shown in Figure 21b, it takes unacceptable 35 seconds to visualize a time series of 20 million points in Grafana for instant inspection by domain experts, and fails to perform over even larger data sizes with out-of-memory error.

In exploratory data analysis, our proposed solution is used to visualize the time series for mining time-varying patterns. For example, by comparing the temperature curves with the widths of track plate crack in railway, the data analysts find their correlations, i.e., higher temperatures generally having narrower widths of track plate crack. It is worth noting that conducting such analysis directly in Python is extremely slow, taking more than 35 seconds for 100 million points, as illustrated in Figure 21b. In contrast, our M4-LSM solutions take only 1.3 seconds, 28 times faster, enabling interactive analysis in practice.

To better understand the challenges of visualizing large scale time series, and inspire further studies in the field, we decide to publish a real dataset of time series with 380 million points from our industrial partner CRRC [https://www.kaggle.com/datasets/leirui1/urban-rail-transit-sensor-dataset]. It contains time series data collected by voltage, vibration, and humidity sensors on Shanghai subway trains.