A PROOFS

A.1 Proof of Proposition 4.1

Proof. For $G = FP$ (or $G = LP$), as defined in Section 4.1, P_G is the point with both the minimum time (or the maximum time) and the largest version number. Therefore, no other chunk in $\mathbb{C}''$ with a version number larger than $P_G.\kappa$ contains a point with the same time as P_G. In other words, P_G is never updated by later appended chunks, having $P_G.t \notin C^{\kappa_1}$ for any $C^{\kappa_1} \in \mathbb{C}''$ with $P_G.\kappa < \kappa_1$. Moreover, P_G is not covered by the delete time range of any delete in $\mathbb{D}$ with a larger version number than $P_G.\kappa$, i.e., $P_G.t \notin D^{\kappa}, D^{\kappa} \in \mathbb{D}, \kappa > P_G.\kappa$. Referring to Formula 2 in Definition 2.7, P_G is a point in the merged time series $M(\mathbb{C}'', \mathbb{D})$, i.e., the latest.

Next we prove that the latest candidate point is the representation result for $G \in \{FP, LP\}$. From Definition 2.7, we know that if P_G is the latest, P_G is a point in $T_i = M(\mathbb{C}'', \mathbb{D})$. Therefore, T_i can be divided into three disjoint subsequences based on P_G,

$$T_l = \{P \mid P \in T_i, P.t < P_G.t\},$$
$$T_m = \{P_G\},$$
$$T_r = \{P \mid P \in T_i, P.t > P_G.t\}.$$

From the distributive property [20] of G, we have

$$G(T_i) = G(T_l \cup T_m \cup T_r) = G(\{G(T_l), G(T_m), G(T_r)\}) = G(\{G(T_l), P_G, G(T_r)\}).$$

It is easy to know that the chunk metadata bounds the time and value range of all points in the chunk. Since P_G is extracted from the boundary points among all chunk metadata, we have

$$P_G.t \leq G(T_i).t \leq \min(G(T_l).t, G(T_r).t), G = FP$$
$$P_G.t \geq G(T_i).t \geq \max(G(T_l).t, G(T_r).t), G = LP$$

i.e., $G(T_i) = G(\{G(T_l), P_G, G(T_r)\}) = P_G$, for $G \in \{FP, LP\}$. □

A.2 Proof of Proposition 4.3

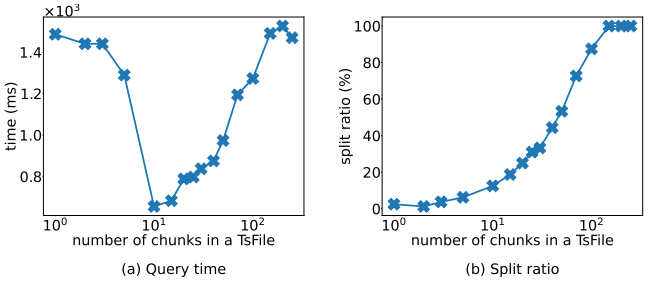
Proof. The first condition states that P_G is not updated by any later appended chunks, having $P_G.t \notin C^{\kappa}$ for any $C^{\kappa} \in \mathbb{C}''$ with $\kappa > P_G.\kappa$. The second condition states that P_G is not covered by any delete in $\mathbb{D}$ with a larger version number than $P_G.\kappa$, i.e., $P_G.t \notin D^{\kappa}$ for any $D^{\kappa} \in \mathbb{D}$ with $\kappa > P_G.\kappa$. Referring to Formula 2 in Definition 2.7, P_G is a point in the merged time series $M(\mathbb{C}'', \mathbb{D})$, i.e., the latest. With the latest candidate point, the proof of the representation result follows the same line of Proposition 4.1. □

B USE CASES

Steel Manufacturing. In a steel manufacturer, 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 reduced to improve energy efficiency.

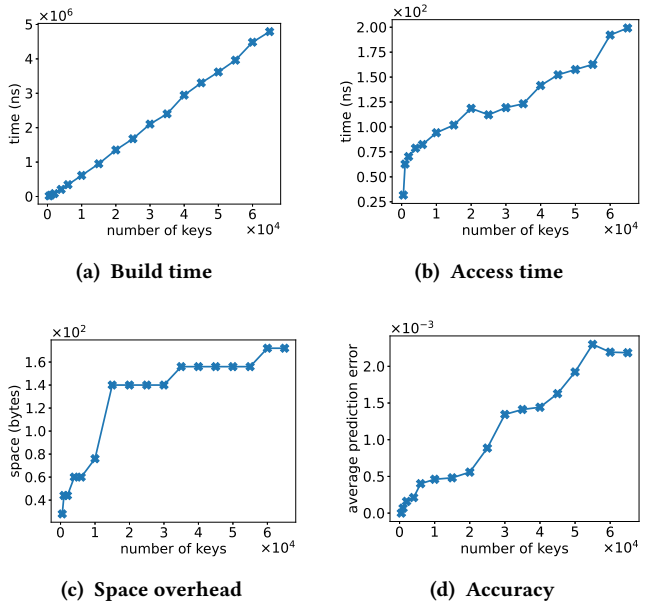
Aviation Industry. Apache IoTDB is employed by a company in the aviation industry to store the high frequency device test data at nanosecond level. Using the IEEE 1588 Precision Time Protocol (PTP), the data collection frequency is as high as 20kHz to 400kHz, generating more than 1T time series in each 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.

Cloud Service. Apache IoTDB is used as the time series database in an Application Performance Monitoring (APM) product. It monitors multiple metrics of the hosts in every 20 milliseconds to 10 seconds, including CPU and network utilization, 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.



(a) Query time (b) Split ratio

Figure 21: M4 query time with varying TsFile size



(a) Build time **(b) Access time**

(c) Space overhead **(d) Accuracy**

Figure 22: Evaluation of time index with step regression

B.1 Experiments on Inter-Chunk Pruning

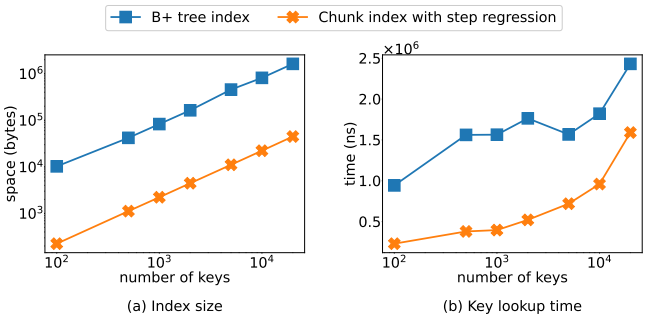
remove?

B.1.1 *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 21. 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 21(a). On the other hand, with the increase of the number of chunks in a TsFile, as illustrated in Figure 21(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 21(a). In this sense, a moderate size of TsFile can improve the query performance by its meta-info across multiple chunks.

B.2 Experiments on Intra-Chunk Indexing

B.2.1 Evaluation of the Proposed Time Index. We evaluate the performance of our proposed time index in terms of build time, access time, space overhead, and accuracy. The experimental results on BallSpeed dataset are shown in Figure 22.



(a) Index size (b) Key lookup time

Figure 23: Comparison with B+ tree index

Build Time. As shown in Figure 22(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.

Access Time. As shown in Figure 22(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 22(c).

Space Overhead. As shown in Figure 22(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.

Accuracy. As shown in Figure 22(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 22(c), the corresponding error increases in Figure 22(d). The reason is that more keys share the same segment leading to higher accumulated error.

B.2.2 Comparison With Baseline Index. We compare the performance of our proposed time index with the traditional B+ tree index in terms of disk space consumption and key lookup time. The experimental results are reported in Figure 23. On average, our time 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 time index with step regression only needs to store the first timestamp of each segment. The better key lookup performance of the time index also benefits from its significantly lower space overhead.