Table A.1: Notations used in the proofs

Notations	Descriptions
N	Number of time-series
N_s	Number of data source
N_m	Number of metric
T	Average number of labels per time-series
S_p	Average size per posting list entry
S_l	Average size per label
S_h	Average size per single hash entry
S_g	Average number of timeseries in a group
T_u	Average number of unique labels per group
T_g	Average number of group labels per group

A Analysis of Metadata storage cost

In this section, we provide formal proofs for the theorems and corollaries presented in Section 3.2.4. Key notations used in the following proofs are summarized in Table A.1.

A.1 Proof of Theorem 3.1

The metadata storage overhead of TSMS (denoted as *Cost*) consists of four components: ❶ the label storage cost within all time-series, denoted as $Cost_{label}$; ❷ the backward index storage cost, denoted as $Cost_{backward}$; ❸ the forward index storage cost, denoted as $Cost_{forward}$; and ❹ the inverted index storage cost, denoted as $Cost_{inverted}$. Therefore, *Cost* can be expressed as:

$$Cost = Cost_{label} + Cost_{backward} + Cost_{forward} + Cost_{inverted} \quad (1)$$

For Prometheus tsdb [7], each time-series object (*i.e., MemSeries*) stores T labels. A bidirectional mapping between every *MemSeries* and its **SeriesID** is maintained in both the backward index and the forward index, while the inverted index keeps, for every label, the complete list of corresponding *SeriesIDs* in a *posting list.* Therefore, the storage cost for the four components of its metadata is as follows:

$$\begin{cases} Cost_{label}(tsdb) = N \cdot T \cdot S_l \\ Cost_{backward}(tsdb) = N \cdot S_h \\ Cost_{forward}(tsdb) = N \cdot S_h \\ Cost_{inverted}(tsdb) = N \cdot T \cdot S_p \end{cases} \quad (2)$$

By combining Equation 1 and Equation 2, we can obtain the expression for $Cost_{tsdb}$:

$$Cost_{tsdb} = N \cdot (T \cdot S_l + 2S_h + T \cdot S_p) \quad (3)$$

For TempusCStore, it adopts a two-dimensional metadata model (see Section 3.2.1 for details). It stores only one metric label within each *MemSeries*, while the $(T-1)$ source labels are stored separately to avoid redundancy. Therefore, $Cost_{label}(TSC)$ is:

$$Cost_{label}(TSC) = N \cdot S_l + N_s \cdot (T-1) \cdot S_l \quad (4)$$

TempusCStore's *backward index* can be divided into the *metric backward index* and the *label backward index*: the former maps a metric label to its *MetricID*, while the latter maps a source label to its *SourceID*. Therefore, the storage cost of the *backward index*, $Cost_{backward}(TSC)$, can be expressed as

$$Cost_{backward}(TSC) = (N_m + N_s) \cdot S_h \quad (5)$$

The forward index of TempusCStore a *MetricID-SourceID* pair to its corresponding *MemSeries*, and the number of index entries equals the number of time-series, *i.e.,* N. Therefore, $Cost_{forward}(TSC)$ is:

$$Cost_{forward}(TSC) = N \cdot S_h \quad (6)$$

The inverted index of TempusCStore stores only *source labels* along with their posting lists of *SourceIDs*. Since each time-series contains $(T-1)$ *source labels* on average, $Cost_{inverted}(TSC)$ is:

$$Cost_{inverted}(TSC) = N_s \cdot (T-1) \cdot S_p \quad (7)$$

By combining Equation 1, Equation 4, Equation 5, Equation 6, and Equation 7, we obtain $Cost_{TSC}$ as:

$$Cost_{TSC} = N \cdot S_l + (N + N_m + N_s) \cdot S_h + N_s \cdot (T-1) \cdot (S_l + S_p) \quad (8)$$

For TimeUnion [50], we examine its group-based model, which clusters multiple time-series into a single group and stores the labels common to all time-series within that group only once. Combining Equation 1, the four components of TimeUnion's metadata storage cost can be expressed as:

$$\begin{cases} Cost_{label}(TU) = \dfrac{N}{S_g} \cdot T_g \cdot S_l + (T - T_g) \cdot N \cdot S_l \\ Cost_{backward}(TU) = \dfrac{N}{S_g} \cdot S_h + N \cdot (T - T_g) \cdot S_h \\ Cost_{forward}(TU) = N \cdot S_h \\ Cost_{inverted}(TU) = \dfrac{N}{S_g} \cdot T_u \cdot S_p + N \cdot (T - T_g) \cdot S_p \end{cases} \quad (9)$$

Here, $\frac{N}{S_g}$ denotes the number of groups. In practice, TimeUnion groups all time-series originating from the same data source into one group; for instance, in the TSBS DevOps dataset, it places the ten time-series that record ten different metrics for the same host into one group [50]. Thus, $\frac{N}{S_g} = N_s$ and $T_g = (T-1)$. Therefore, the four components of TimeUnion's metadata storage overhead can be expressed as:

$$\begin{cases} Cost_{label}(TU) = N_s \cdot (T-1) \cdot S_l + N \cdot S_l \\ Cost_{backward}(TU) = N_s \cdot S_h + N \cdot S_h \\ Cost_{forward}(TU) = N \cdot S_h \\ Cost_{inverted}(TU) = N_s \cdot T_u \cdot S_p + N \cdot S_p \end{cases} \quad (10)$$

Combining Equation 1 and Equation 10, the metadata storage cost of TimeUnion, $Cost_{TU}$, is:

$$Cost_{TU} = (N_s \cdot (T-1) + N) \cdot S_l + (2N + N_s) \cdot S_h + (N_s \cdot T_u + N) \cdot S_p \quad (11)$$

In summary, we have proven Theorem 3.1.

A.2 Proof of Corollary 3.2

We denote the difference in metadata storage cost between two TSMS systems, A and B, as $SaveCost(A, B)$, which equals $Cost_A - Cost_B$.

By incorporating Equation 2, $SaveCost(A, B)$ can be decomposed into the following four components:

$$\begin{cases} SaveCost_{label}(A,B) = Cost_{label}(A) - Cost_{label}(B) \\ SaveCost_{backward}(A,B) = Cost_{backward}(A) - Cost_{backward}(B) \\ SaveCost_{forward}(A,B) = Cost_{forward}(A) - Cost_{forward}(B) \\ SaveCost_{inverted}(A,B) = Cost_{inverted}(A) - Cost_{inverted}(B) \end{cases} \quad (12)$$

Combining Equation 12, Equation 2, and Equation 4-7, the difference in metadata storage cost between TempusCStore and Prometheus tsdb is as follows:

$$\begin{cases} SaveCost_{label}(TCS, tsdb) = (T-1) \cdot (N - N_s) \cdot S_l \\ SaveCost_{backward}(TCS, tsdb) = (N - N_m - N_s) \cdot S_h \\ SaveCost_{forward}(TCS, tsdb) = 0 \\ SaveCost_{inverted}(TCS, tsdb) = (N \cdot T - N_s \cdot (T-1)) \cdot S_p \end{cases} \quad (13)$$

Since $T \geq 1$ and $N \geq N_s$ (because $N_s = \frac{N}{N_m}$ and $N_m \geq 1$), $SaveCost_{label}(TCS, tsdb) \geq 0$. When $N_m \geq 2$ and $N_s \geq 2$, $SaveCost_{backward}(TCS, tsdb) \geq 0$. Since $(N \cdot T) \geq (N_s) \cdot (T-1) \geq 0$, $SaveCost_{inverted}(TCS, tsdb) \geq 0$. Therefore, the following corollary can be derived.

$$Cost_{TCS} \leq Cost_{tsdb} \quad \text{if} \quad N_m \geq 2 \quad \text{and} \quad N_s \geq 2 \quad (14)$$

Combining Equation 12, Equation 10, and Equation 4-7, the difference in metadata storage cost between TempusCStore and TimeUnion is as follows:

$$\begin{cases} SaveCost_{label}(TCS, TU) = 0 \\ SaveCost_{backward}(TCS, TU) = (N - N_m) \cdot S_h \\ SaveCost_{forward}(TCS, TU) = 0 \\ SaveCost_{inverted}(TCS, TU) = N_s \cdot S_p \cdot (T_u - T + 1) + N \cdot S_p \end{cases} \quad (15)$$

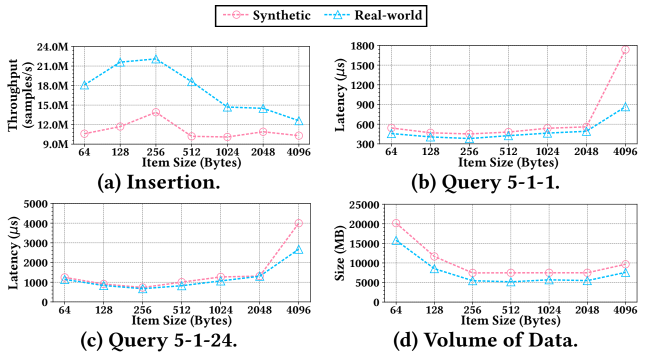

Figure B.1: Impact of item size.

Since $N \geq N_m$, $SaveCost_{backward}(TCS, TU) \geq 0$. T_u is the number of unique labels obtained by deduplication of all labels in T_g time-series. Since the labels in different time-series are not exactly the same, $T_u > T$. Therefore, $SaveCost_{inverted}(TCS, TU) > 0$. Thus, the following corollary can be drawn:

$$Cost_{TCS} \leq Cost_{TU} \quad (16)$$

In summary, we have proven Corollary 3.2.

B Item Size

We evaluated the impact of item size on performance and compression. As Figure B.1 shows, by setting the item size at 256 bytes, TempusCStore achieves the best performance in terms of both insert throughput and query latency. Smaller item sizes lower compression efficiency and raise flush overhead, which shall limit insertion throughput. In spite of that, the query latency can be shortened as decompression costs are reduced. Conversely, too large an item size, while improving compression, increases decompression needs during queries, raising query latency. In TempusCStore, larger item sizes require MemSeries to accumulate more samples in memory before sending them to TreeSeries, disrupting early migration and reducing insertion throughput. To balance these factors, we set the item size to 256 bytes.