APPENDIX

INDEXING ANALYZE

In this subsection, we analyze the indexing cost in terms of index space and query latency, and the symbols involved are shown in Table XIII. First, the total size of the index space is shown in Equation 1, including the $forward\_index$ containing ART and FST and $inverted\_index$.

$$Storage\_Cost = Cost_{forward\_index} + Cost_{inverted\_index} \tag{1}$$

We will explain every part step by step, the first is the inverted

TABLE XIII: Indexing analysis notations.

Symbol	Interpretation
S_s	Average size (bytes) of the series-level posting list
S_g	Average size (bytes) of the group-level posting list
S_p	Average size (bytes) of posting list pointer
N_s	Number of time-series in a Segment
N_t	Average number of tag pairs per time-series
N_e	Average number of entries per posting list
N_d	Number of different primary tags in a Segment
N_q	Number of secondary tag pairs in queries
N_p	Number of primary tags in a Segment
N_k	Number of primary *tagks* in a Segment
L_a	Latency of accessing one node of *art*
L_f	Latency of accessing one node of *fst*
L_g	Latency of updating a group-level posting list
L_s	Latency of updating a series-level posting list
L_{disk}	Latency to access 1 block in a disk
l_{mkv}	Average length (bytes) of $metric + tagk + tagv$
l_k	Average length (bytes) of $tagk$
Q_k	Average number of $tagk$ per Query
Q_t	Average number of tag pairs per Query
S_b	Size of accessing disk basic unit *block*
R_p	Compression ratio for posting list
IO_s	Average number of IO *block* without grouping
IO_g	Average number of IO *block* with grouping
R_n	Ratio of constructing a new group for new series key

index. And the occupied space of posting lists is equivalent to storage cost of inverted index. The cost without grouping is as follows:

$$\begin{aligned} Cost_{inverted\_index1} &= \sum_{i=0}^{tag\_num} (S_s + S_p) \\ &= \frac{N_s \cdot N_t}{N_e} \cdot (S_s + S_p) \\ &= \frac{N_s \cdot N_t}{N_e} \cdot S_s + \frac{N_s \cdot N_t}{N_e} \cdot S_p \end{aligned} \tag{2}$$

In Equation 2, $N_s \cdot N_t / N_e$ indicates how many tag pairs are used as terms of inverted index. We need to store identifiers of each tag (*seriesIDs*), its storage overhead is $N_s \cdot N_t \cdot S_s / N_e$. Also for each time-series, we need to store a pointer to the posting list, so the total size of this part is $N_s \cdot N_t \cdot S_p / N_e$. Based on the grouping support of AdpDM, the costs appear as follows:

$$\begin{aligned} Cost_{inverted\_index2} = &\sum_{i=0}^{primary\_num} (S_g + S_p) \\ &+ \sum_{i=0}^{secondary\_num} (S_s + S_p) \\ &= N_d \cdot (S_g + S_p) + N_q \cdot (S_s + S_p) \end{aligned} \tag{3}$$

The storage overhead mainly includes two parts in Equation 3. One is the group-level inverted index including seriesIDs $N_d \cdot S_g$ and pointers $N_d \cdot S_p$. The number of primary tags selected N_d is far less than the total number of tag pairs $N_s \cdot N_t / N_e$, reduced by 2-3 orders of magnitude. And the length of the posting list at the group level S_g is also less than or equal to the original average length S_s. The other is the overhead of building a series-level inverted index for the queried tag pairs not belonging to primary tags. We find out this type of tag accounts for less than 5% in real-world datasets. It also consists the seriesIDs $N_q \cdot S_s$ and pointers $N_q \cdot S_q$. In summary, the space of the inverted index is greatly reduced after applying primary tags.

The following is an analysis of our forward index. If no grouping strategy is used, the storage overhead is as follows

$$Cost_{forward\_index1} = \frac{N_s \cdot N_t}{N_e} \cdot l_{mkv} \cdot art\_node\_size \tag{4}$$

In Equation 4, we estimate the upper bound of ART by multiplying the node size by the number of nodes, where the number of nodes is calculated by multiplying the total number of tag pairs $N_s \cdot N_t / N_e$ by the average character length l_{mkv}.

$$\begin{aligned} Cost_{forward\_index2} &= S_{art} + S_{fst} + S_{query} \\ &= N_k \cdot l_k \cdot art\_node\_size \\ &+ N_k \cdot FST\_size \\ &+ N_q \cdot Q_k \cdot l_{mkv} \cdot art\_node\_size \end{aligned} \tag{5}$$

In Equation 5, we calculate the total space of our strategy, which includes ART, FST and a few series-level posting lists respectively. Since ART only needs to index tagk in our index. Its upper bound of space cost is $N_k \cdot l_k \cdot art\_node\_size$. The FST is built to search all kinds of tagv which matches the same tagk, so the total size of the FST is $N_k \cdot FST\_size$. As for series-level posting lists in the query, similar to Equation 4, the ART size S_{query} is approximately $N_q \cdot Q_k \cdot l_{mkv} \cdot art\_node\_size$. In order to accurately analyze the difference, we divide the two equations to get Equation 6. Because S_{query} including a very small amount of data should be omitted.

$$\begin{aligned} \frac{Cost_{forward\_index2}}{Cost_{forward\_index1}} &= \frac{N_k \cdot (l_k \cdot art\_node\_size + FST\_size)}{\frac{N_s \cdot N_t}{N_e} \cdot l_{mkv} \cdot art\_node\_size} \\ &= \frac{N_k}{\frac{N_s \cdot N_t}{N_e}} \cdot \frac{l_k \cdot art\_node\_size + FST\_size}{l_{mkv} \cdot art\_node\_size} \\ &= A \cdot B \end{aligned} \tag{6}$$

In Equation 6, A represents the ratio of the number of primary tags and the number of all tag pairs in the segment, which is a small number. And B is expressed as the ratio of the unit space overhead of two indexes, which is a constant close to 1. For instance, this is true for *Infras* dataset as A = 1.48x10^{-3}, B = 0.95. Therefore, our forward index scheme is better than the existing methods.

Second, we analyze query latency in our index and compare it with other indexings. In Equation 7, each overhead of the query latency is illustrated. Traversing forward indexes for pointers of the posting list is the first part of time overhead $L_{forward\_index}$. Next, $L_{inverted\_index}$ consists of extracting the posting lists from the disk and intersecting them to determine the final return seriesIDs. Finally, L_{data_IO} represents the latency of reading series keys in the disk.

$$Query\_Lantency = L_{forward\_index} + L_{inverted\_index} + L_{data\_IO} \tag{7}$$

First is the delay analysis of the forward index, the expression without primary tags is as follows. In the formula, we only need to deduce the latency of querying a single tag, and then multiply it by Q_t

$$\begin{aligned} L_{forward\_index1} &= log(art\_node\_num) \cdot L_a \cdot Q_t \\ &= log(N_s \cdot N_t \cdot l_{mkv}) \cdot L_a \cdot Q_t \end{aligned} \tag{8}$$

In Equation 8, since the searching complexity of the radix tree is $log(art\_node\_num)$, the worst space overhead is specifically expressed as $N_s \cdot N_t \cdot l_{mkv}$. In our scheme, the cost is shown as follows:

$$\begin{aligned} L_{forward\_index2} &= L_{ART} + L_{FST} \\ &= log(N_k \cdot l_k) \cdot L_a \cdot Q_t + L_f \cdot Q_t \end{aligned} \tag{9}$$

In Equation 9, L_{ART} is similar to the above analysis. The number of nodes is estimated to be the product of the tagk number N_k and the string length l_k. The time cost of FST is basically expressed as searching the Q_t numbers of search time L_f in FST. In order to compare the gap of the forward index in detail, we make the difference *D* between the two formulas to get Equation 10:

$$D = [log(\frac{N_s \cdot N_t \cdot l_{mkv}}{N_k \cdot l_k}) \cdot L_a - L_f] \cdot Q_t \tag{10}$$

First, the value range of $log(\frac{N_s \cdot N_t \cdot l_{mkv}}{N_k \cdot l_k})$ is between 5-7, which belongs to the constant order. Second, L_a and L_f are of the same order of magnitude, so the final result D is close to 0. Therefore, after introducing FST, the storage overhead of the forward index is still close to the original scheme. The cost of inverted indexes is stated as followed:

$$L_{inverted\_index1} = Q_t \cdot \lceil \frac{R_p \cdot S_s}{S_b} \rceil \cdot L_{disk} + L_{intersection1} \tag{11}$$

$$L_{inverted\_index2} = Q_t \cdot \lceil \frac{R_p \cdot S_g}{S_b} \rceil \cdot L_{disk} + L_{intersection2} \tag{12}$$

$L_{inverted\_index}$ includes two parts. First, the latency of extracting the posting lists is expressed as the number of I/Os multiplied by the disk IO latency L_{disk}. In Equation 11, the number of I/Os is equal to the number of data blocks $Q_t \cdot (R_p \cdot S_s/S_b)$ of the posting list involved in the queried tag pairs. As for the overhead calculation after grouping, it is shown in Equation 12. The only difference is the size of the posting list S_s at the series level is replaced by the group level S_g. The second part is the intersection of multiple posting lists. Because the average length of lists is proportional to intersection overhead, lists with primary tags have an advantage over series-level inverted index.

$$L_{data\_IO1} = Q_t \cdot IO_s \cdot L_{disk} \tag{13}$$

$$L_{data\_IO2} = Q_t \cdot IO_g \cdot L_{disk} \tag{14}$$

Finally, the I/O latency of the series keys in Equations 13 and 14, is multiplied by the number of I/Os ($Q_t \cdot IO_s$ and $Q_t \cdot IO_g$) and the disk I/O latency L_{disk}. After the classification by group, IO_g will be much smaller than IO_s, so after the query is hit, data can be obtained with the least I/O to ensure query efficiency.

THROUGHPUT ANALYZE

Next, we analyze the throughput of real-time index building, and the symbols involved are also shown in Table XIII. A key factor is the overhead of updating the inverted indexes. In our algorithm, the overhead of adding a new group or correcting posting list equals $N_p/N_s \cdot L_g$, which is the product of the average number of primary tags and the group-level latency, and for the series-level inverted index is $N_t \cdot L_s$. We analyze the total cost of updating the inverted index for baseline and AdpDM in Equation 15 and Equation 16. The $total\_num$ means the total amount of series keys ingestion.

$$Cost\_update1(total\_num) = N_t \cdot L_s \cdot total\_num \tag{15}$$

$$Cost\_update2(total\_num) = \frac{N_p}{N_s} \cdot L_g \cdot R_n \cdot total\_num \tag{16}$$

The specific quantification method is to multiply the cost of a single update by the number of required operations. The total operation numbers of AdpDM can be estimated as the ratio of constructing a new group multiplied R_n by the total num of series keys However, in other TSDBs, it is necessary to update index for each series key. In fact, R_n is about 5% and N_p/N_s is less N_t. Therefore, the time cost of AdpDM is far superior to baselines in theory.