Randomized Sketches for Quantile in LSM-tree based Store

ZILING CHEN, Tsinghua University, China
SHAOXU SONG*, Tsinghua University, China

Quantiles are costly to compute exactly but can be efficiently estimated by quantile sketches. Extensive works on summarizing streaming data, such as KLL sketch, focus on minimizing the cost in memory to provide certain error guarantees. For the problem of quantile estimation of values in LSM-tree based stores, streaming methods have an expensive I/O cost linear to data size N. Since disk components (chunks and SSTables) in the LSM-tree are immutable once flushed, quantile sketches can be pre-computed as a type of statistics to reduce I/O cost and accelerate queries. Unfortunately, to provide deterministic additive ϵN error guarantees on queried data, all pre-computed deterministic sketches of queried chunks each with size N_c should provide ϵN_c error guarantee, resulting in no improvement in the linear I/O cost. In this study, we propose pre-computing randomized sketches which provide randomized additive error guarantees. Our major technical contributions include (1) randomized sketches for data chunks constructed in flush events, which are proved to be optimal and achieve an I/O cost proportional to $\sqrt{N}$, (2) hierarchical randomized sketches for SSTables constructed in compaction events, that further improve the asymptotic I/O cost, (3) the KLL sketch summarizing proposed pre-computed sketches is proved to be more accurate than that summarizing streaming data, and proved to achieve sublinear I/O cost while achieving the same memory complexity as in the streaming settings. Extensive experiments on synthetic and real datasets demonstrate the superiority of the proposed techniques. The approach is deployed in an LSM-tree based time-series database Apache IoTDB.

CCS Concepts: • **Information systems** → *DBMS engine architectures*; • **Theory of computation** → **Sketching and sampling**.

Additional Key Words and Phrases: Quantile, LSM storage, Database statistics, Sketch

ACM Reference Format:
Ziling Chen and Shaoxu Song. 2025. Randomized Sketches for Quantile in LSM-tree based Store. *Proc. ACM Manag. Data* 3, 1 (SIGMOD), Article 67 (February 2025), 26 pages. https://doi.org/10.1145/3709717

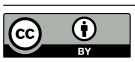
1 Introduction

Quantiles are order statistics that characterize the distribution of data [6, 40]. Common quantiles include median, quartiles and percentiles. Quantile computation is fundamental in data science tasks including data profiling [19], monitoring [32] and outlier detection [4]. Computing exact quantiles is costly, either loading all data in memory or scanning data in multiple rounds [33]. Thus, quantiles are often estimated by data structures named quantile sketches summarizing the whole dataset.

Formally, given a set of values $\{x_1, \ldots, x_N\}$ equipped with a total order, the rank of value y is $R(y) = |\{x_i \mid x_i \leq y\}|$, and the ϕ-quantile is the value y having $R(y) = \phi N$. Quantile sketches summarize data arriving in a streaming fashion and provide estimated ranks and quantiles. Most works [6, 18, 27, 31] focus on providing additive ϵN error guarantee, i.e., $\left|\hat{R}(y) - R(y)\right| \leq \epsilon N$ for

*Shaoxu Song (https://sxsong.github.io/) is the corresponding author.

Authors' Contact Information: Ziling Chen, Tsinghua University, China, chen-zl22@mails.tsinghua.edu.cn; Shaoxu Song, Tsinghua University, China, sxsong@tsinghua.edu.cn.



ACM 2836-6573/2025/2-ART67
https://doi.org/10.1145/3709717

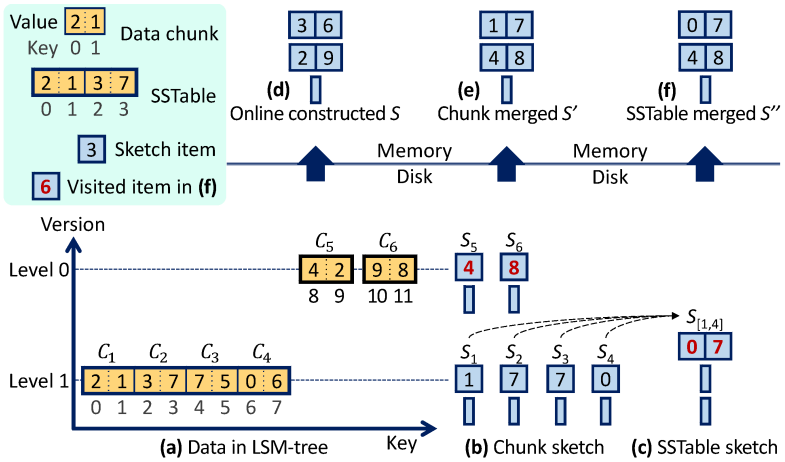


Fig. 1. Overview of quantile estimation on values in LSM-tree based KV store with pre-computed sketches, including (a) data in LSM-based KV store, (b) sketches for chunks in Section 3.1, (c) hierarchical sketches for SSTables in Section 3.2, (d) query processor summarizing data online with a sketch in memory, (e)(f) query processor summarizing pre-computed sketches in Section 4.

any value y, and analyze the required space in memory. In recent years, a family of randomized sketches analyze the randomized guarantee of $\Pr\left[\left|\hat{R}(y)-R(y)\right|\leq \epsilon N\right]\geq 1-\delta$ to improve the space bound.

The log-structured merge-tree (LSM-tree) [34] is widely adopted to handle intensive writes, e.g., in Apache IoTDB [37]. In the LSM-tree, incoming data are buffered in memory and flushed to disk as sorted-string tables (SSTables) at level 0. Compaction events are triggered to merge SSTables into higher levels.

Pre-computed statistics in LSM-tree can improve query performance by reducing I/O. For the lookup performance on specific keys, statistics like Bloom filters [3] and extreme keys of chunks [24, 28] can be utilized. For the performance of aggregating values, extreme values and counts are kept in IoTDB [24], while histograms and wavelets are kept for cardinality estimation in [1]. In this paper, we consider the quantile estimation of values in LSM-based KV stores, with the help of pre-computed quantile sketches.

1.1 Quantile Sketch in LSM-tree Store

For the data model, each node in the LSM-tree is a data file in SSTable format on disk, denoted by dashed rectangles in Figure 1(a). The SSTable file is immutable, consisting of key-value pairs ordered by key, and partitioned into size-fixed data chunks (also named pages or blocks) [17, 24, 29, 30] denoted by solid rectangles with size $N_c = 2$ in Figure 1(a). Ingested data results in new SSTables at level 0. In the LSM-tree with size-tiered compaction policy, T nodes occurred at the same level L will be merged into a node at level $L+1$. For the example in Figure 1(a) where $T=4$, the SSTable at level 1 comes from merging 4 chunks.

Existing quantile query processing is online-sketching the data fetched from LSM-tree. Data points read into memory are summarized by a quantile sketch as in Figure 1(d). It is implemented in systems like InfluxDB [23] and needs expensive linear I/O cost.

The goal of the pre-computation is to attach quantile summaries to each data chunk (size-fixed partition) and SSTable (the node). As in Figure 1(b)(c), chunk summaries S_i are built from chunk data C_i, while SSTable summaries $S_{[l,r]}$ are built from chunk summaries S_i.

For the query model, the quantile query tries to combine information from summaries for SSTables or chunks fully contained in the query, and some data points in the left over parts. Figure 1 shows that to process a quantile query posed on all data, the query processing (f) reads sketch items in $S_5, S_6, S_{[1,4]}$ from disk and does not need to visit all data.

As for the constraints in queries, the disk-bounded solutions in Section 3 load all pre-computed sketches in memory, i.e., it is not as space constrained as the streaming model. For the memory-constrained solutions in Section 4, the memory budget is limited as KLL in streaming model, i.e., the data structure maintained in memory should be small. Thereby, the pre-computed summaries are further compressed in memory.

Although any fully-mergeable [2] quantile sketch can be applied, whether randomized or not, the key problem is to provide the aforementioned ϵN error guarantee on N queried data. The properties of randomized sketches and randomized error guarantees promise an asymptotic improvement in I/O cost, even with keeping a memory bound as in streaming settings. Intuitions are as follows.

1.2 Intuition

Since chunks and SSTables in LSM-trees are immutable, pre-computed quantile sketches can be constructed in the insertion-only model, i.e., they do not need to consider deletes or updates. Sketches are built when flush events and compaction events happen. For the chunk sketch, all the data in the chunk are accessible in memory when the chunk is being flushed to disk. While for the SSTable, not all data are in memory during compactions because of the partitioning. Thus, SSTable sketches are built based on chunk sketches as in Figure 1(b)(c) to avoid costly asynchronous scans.

Now we discuss why randomized quantile sketches in LSM-trees can improve asymptotic I/O cost. To provide ϵN error guarantee on N queried data, all pre-computed deterministic sketches have to achieve the same ϵ, such as when data are independently identical distributed (i.i.d.). That leads to trivial results of I/O bounds linear to N. However, when the target is to provide ϵN error guarantee with a high probability of $1-\delta$, each pre-computed randomized sketch does not need to achieve that error guarantee. For example, when a chunk sketch guarantees $\Pr\left[\left|\hat{R}(y)-R(y)\right| \leq \epsilon N_c\right] \geq 0.99$, k such sketches together could guarantee $\left|\hat{R}(y)-R(y)\right| \leq k\epsilon N_c$ with confidence much higher than 0.99. In other words, for fixed ϵ, δ, as N grows and more chunk sketches queried, each sketch can be less accurate and smaller in size. That promises a total I/O cost sublinear to N. This property may be hard to apply in chunks as sketches are built ahead of query, but can be applied in SSTables as higher level SSTables contains more data. That is, SSTable sketches can properly compress chunk sketches for asymptotic improvements in I/O.

So far we simply load pre-computed sketches into memory in the query process, i.e., the memory cost is equal to the I/O cost. The cost is better than linearity, but still larger than the memory bounds of existing sketches constructed from streaming data. When the memory budget is constrained, we can treat pre-computed sketches as weighted inputs of existing streaming algorithms like the KLL sketch [27], resulting in a size-limited sketch in memory. Since this memory-constrained solution introduces error during compressing pre-computed sketches, the sketches should be more accurate to keep the overall ϵN error guarantee.

1.3 Challenge

The essential problem is to pre-compute randomized sketches that can achieve asymptotic improvements in I/O cost. For chunk sketch construction, the key is to leverage the fact that all data are in memory, which is different from streaming settings. The challenge is to obtain the optimal

result in the randomized sketch format. For SSTable sketch construction, it should properly compress chunk sketches and still be effective when the SSTable is large and not all data are queried in the query with specific key ranges. Key analyses consist of the improved I/O cost with chunk sketches and SSTable sketches, as well as whether the improvement exists when the memory is constrained.

1.4 Contributions

Our major contributions in this study are as follows.

(1) The chunk sketch is built holistically from the entire chunk data in memory, which is different from streaming methods such as KLL that summarize data one by one. Thereby, the chunk sketch is proved to be the optimal randomized sketch in Proposition 3.3 and improves the I/O complexity in Proposition 3.4.

(2) The hierarchical sketches for the SSTable are built from compressing chunk sketches and effective when only a part of chunks in the SSTable are queried. It is different from summarizing all data with a single KLL sketch, which cannot accelerate quantile queries on partial data. The SSTable sketches have bounded error in Proposition 3.6 and further improve I/O complexity in Proposition 3.7.

(3) Our analyses show that when the memory budget in query is constrained, to guarantee the same ϵN error, compressing pre-computed sketches has a better I/O complexity than KLL summarizing data fetched in a streaming fashion, as presented in Propositions 4.3, 4.4 and Table 3.

(4) The proposed method has become a function in an LSM-tree based time-series database Apache IoTDB [24]. The document is released in the product website [21]. The code is included in the GitHub repository of IoTDB by the system developers [22].

(5) Extensive experiments on synthetic and real datasets demonstrate the superiority of our proposals. The required I/O cost to achieve ϵN error guarantee is shown to be consistent with our analyses. The time cost of the query process is significantly improved. The related code and data are available for reproducibility [5].

2 Preliminaries

Here we introduce preliminaries of randomized quantile sketches and LSM-based stores. Table 1 lists the frequently used notations.

2.1 Randomized Quantile Sketch

The family of quantile sketches [2, 27, 31] summarize data with a structure S consisting of H leveled compactors. The compactor at height h, denoted as $S[h]$, stores items with the weight of 2^{h-1}, where $1 \leq h \leq H$. For any value y, its estimated rank $\hat{R}(y)$ in the sketch is the sum of its weighted ranks in all compactors.

Arriving data are first inserted into the bottom compactor. Each compactor has capacity and triggers a compaction operation when full. A compaction will sort items, output even-indexed or odd-indexed items to the next compactor, and clear the current one. Figure 2 shows an example of adding values $\{5, 9, 3, 6, 6\}$ to a KLL sketch [27]. The first compaction keeps the even-indexed 9 in $\{5, 9\}$ and the second compaction keeps the odd-indexed 3 in $\{3, 6\}$, resulting in Figure 2(c). The estimated rank of the value 7 is $\hat{R}(7) = 1 * 2^0 + 1 * 2^1 = 3$ and the error is $\hat{R}(7) - R(7) = 3 - 4 = -1$.

Randomized compactions, first introduced by Agarwal et al. [2] bring internal randomness. A randomized compaction outputs odd or even indexed items randomly with equal probability, not always odd, always even, or taking turns. Thus, the example in Figure 2 is a possible case after

Table 1. Notations

Symbol	Description
N	Number of data points queried
S	Sketch in memory during query execution
$R(x)$	Real rank of value x in queried data
$\hat{R}(x)$	Estimated rank of value x
H	Height of sketch S
$S[h]$	Compactor at height h in sketch S
m_h	Number of compactions at height h
K	Capacity of the top compactor $S[H]$
γ	Capacity decrement factor in sketch S
C_i	The i-th data chunk in LSM-tree
N_c	Number of data points in each data chunk
S_i	Chunk sketch for C_i
H_c	Height of each chunk sketch
M_c	Size limit of each chunk sketch
T	Size ratio of components in an LSM-tree
T_s	Size ratio of sketches in an SSTable

two randomized compactions. Figure 3 shows the impact of a randomized compaction on rank estimation. The error introduced by a randomized compaction at height h, is a random variable $\in \{-2^{h-1}, 0, 2^{h-1}\}$ which is zero-mean sub-Gaussian. A zero-mean variable X with variance σ^2 is sub-Gaussian if $\mathbb{E}[\exp(sX)] \leq \exp(-\frac{1}{2}s^2\sigma^2)$ for any $s \in \mathbb{R}$.

The error of the randomized sketch, $\hat{R}(x) - R(x)$, is the sum of these independent sub-Gaussian variables. Let m_h be the number of compactions at height h, $\hat{R}(x) - R(x)$ is zero-mean sub-Gaussian with a variance $\sigma^2 \leq \sum_{h=1}^{h=H-1} 4^{h-1} m_h$ as in [2, 27]. Then we apply the Chernoff tail bound for sub-Gaussian variables: $\Pr\left[|\hat{R}(x) - R(x)| > a\right] \leq 2\exp\left(-\frac{a^2}{2\sigma^2}\right)$, for any $a>0$. Thus, the randomized guarantee of $\Pr\left[\left|\hat{R}(y) - R(y)\right| \leq \epsilon N\right] \geq 1-\delta$ holds, when $\sigma^2 \leq \frac{1}{2}\epsilon^2 N^2 / \log\frac{2}{\delta}$. Since $\sigma^2 \leq \sum 4^{h-1} m_h$ as just introduced, the number of randomized compactions m_h should be small enough. This is the basis of our I/O cost analyses.

KLL sketch [27] is the randomized sketch achieving the best asymptotic behavior in the streaming model. It takes streaming data as input and summarizes them with leveled compactors and randomized compactions as just introduced. The main conceptual contribution of KLL is the exponentially decreasing capacities for compactors. In detail, the parameter K is the capacity of the top compactor, and $\gamma \in (\frac{1}{2}, 1)$ is the capacity decrement factor. The capacity of the compactor at height h is set to be $K\gamma^{H-h}$. That results in exponentially increasing compactions at lower heights and provides the optimal bound of variance of the sub-Gaussian error. A mergeable version in the open-source algorithms library Apache Datasketches [12] achieves a memory complexity of $O\left(\frac{1}{\epsilon}(\log\frac{1}{\delta})^{\frac{1}{2}}\right)$ to provide ϵN error guarantee on N data with probability at least $1-\delta$. Its I/O complexity is $O(N)$ for the streaming input.

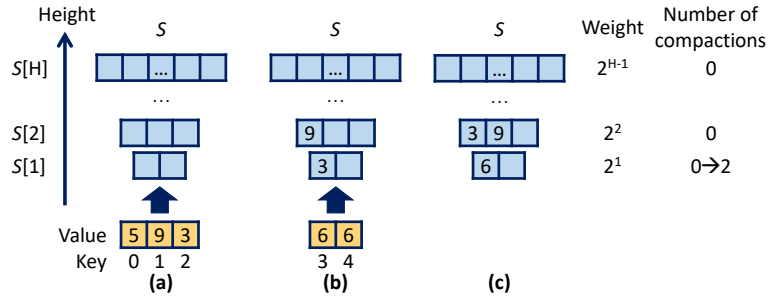


Fig. 2. Process of feeding values to a sketch with leveled compactors. The first compaction keeps even-indexed 9 in $\{5, 9\}$. The second compaction keeps odd-indexed 3 in $\{3, 6\}$.

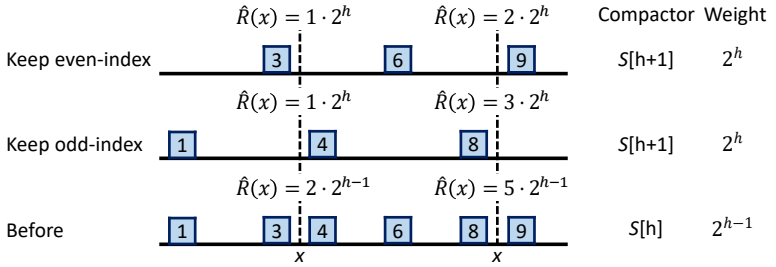


Fig. 3. A randomized compaction over 6 items at height h.

2.2 LSM-Tree based Store

In LSM-tree based systems, a batch of flushed data are stored as an SSTable at level 0 and then compacted (merged) to higher level SSTables. The size ratio T defines the frequency of SSTable compaction and affects the shape of LSM-tree [25].

To optimize the merge process, LSM-tree based systems partition SSTables into several fixed-size chunks or pages [30]. In the widely used tiering policy [13, 14], there are at most $T-1$ SSTables in each level. A level L SSTable is at most T times larger than a SSTable at level $L-1$. SSTables at the same level may have overlapping key ranges [30]. Figure 1(a) shows an LSM-tree with size ratio $T = 4$ and chunk size $N_c = 2$. The SSTable at level 0 has 1 chunk in this case. The SSTable at level 1 is built by merging 4 level 0 SSTables and partitioned into 4 chunks $C_1, \ldots, C_4$.

When performing an aggregate query posed on data in a key range, data in all chunks overlapped with queried key range are read and the values are aggregated one by one. To accelerate this, LSM-tree based systems can pre-compute statistics for SSTables and chunks as in [1, 37].

2.3 Online KLL Sketch Construction over LSM-tree based Store

Quantile estimation in LSM-tree based systems has not been benefited from pre-computed summaries yet. When estimating quantile with KLL algorithm, the KLL sketch in memory is built online with all queried data points one by one, which I/O cost is $O(N)$.

As shown in Figure 1(d), the KLL sketch S in memory is online built with all the 12 data points $\{2, 1, 3, 7, 7, 5, 0, 6, 4, 2, 9, 8\}$ arriving in a streaming fashion. Randomized compactions are triggered during the process.

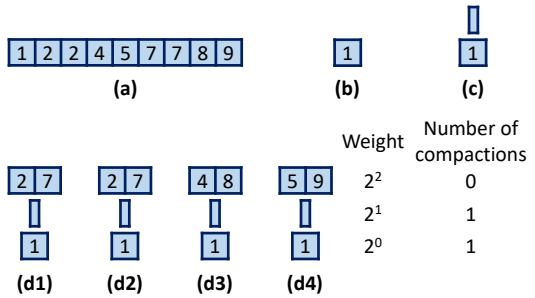


Fig. 4. Example of building a randomized sketch for a chunk with 9 values $\{1, 9, 5, 2, 7, 2, 8, 4, 7\}$ under size limit of 3. (a) sorts all values in memory and (b)(c)(d) assign compactors from bottom to top. (d1)...(d4) show all possible results because of the internal randomness.

3 Randomized Sketches on Disk

We introduce pre-computed sketches for chunks and SSTables in Sections 3.1 and 3.2 respectively. For each type of sketch, we present the construction algorithm during LSM-events, analyze the error, and show the asymptotic I/O cost.

3.1 Optimal Chunk Sketches

The randomized sketch for the chunk is constructed during LSM-tree flushing. Since chunks are basic units of I/O and any chunk to flush is stored in memory, we propose building an optimal sketch with all in-memory data. In the following, we present Algorithm 1 for constructing a chunk sketch, analyze its optimality in Proposition 3.3, and show the I/O cost (total size of chunk sketches) sublinear to N in Proposition 3.4.

3.1.1 Construction Algorithm. Algorithm 1 illustrates the construction of chunk sketch S_i for any chunk C_i during flushing. In the KV-store, data in the chunk are sorted by keys, while the quantile queries in this paper are on values. Thus, the algorithm starts by sorting the values of chunk data cached in memory. Based on the chunk size N_c and the size limit M_c on each chunk sketch, the height H_c of the randomized sketch is computed (Line 2). Then the bottom $H_c - 1$ compactors $S_i[1], \ldots, S_i[H_c - 1]$ are assigned (Line 3-7). Finally the top compactor $S_i[H_c]$ is computed (Line 8).

Algorithm 1: Building optimal randomized sketch in a chunk

Input: Chunk C_i, chunk sketch size limit M_c
Output: The optimal sketch S_i with H_c compactors

1 $v_1, \ldots, v_{N_c} \leftarrow$ the sorted N_c values of data in C_i;
2 $r,\ H_c \leftarrow 1,\ 1 + \lceil \log_2 \frac{N_c}{M_c} \rceil$;
3 **for** $h \leftarrow 1$ **to** $H_c - 1$ **do**
4 **if** $\lfloor N_c / 2^{h-1} \rfloor$ MOD $2 = 0$ **then** $S_i[h] \leftarrow \emptyset$;
5 **else** $S_i[h],\ r \leftarrow \{v_r\},\ r + 2^{h-1}$;
6 $r \leftarrow r + \text{RANDOM}(0, 1) \cdot 2^{h-1}$;
7 **end**
8 $S_i[H_c] \leftarrow \left\{ v_{r+i \cdot 2^{H_c - 1}} \mid 0 \le i < N_c / 2^{H_c - 1} \right\}$;

Example 3.1. Figure 4 shows an example of building a chunk sketch from N_c=9 values $\{1, 9, 5, 2, 7, 2, 8, 4, 7\}$ when M_c=3. The proposed algorithm sorts all values in the chunk at the beginning. The height of the sketch is calculated to be H_c=3 and the three compactors are constructed from the bottom to the top. Figure 4(d1)...(d4) are the result of the randomized variable r being 0...3, respectively.

The built sketch S_i has a T-shape, where only the top compactor $S_i[H_c]$ has more than one item. The randomness of the sketch comes from the random bits in Line 6 generated $H_c - 1$ times.

3.1.2 Error Analysis. Now we discuss the performance of the chunk sketch more formally. Since the error of randomized quantile sketches comes from randomized compactions, we first show the number of compactions in the construction as follows.

Lemma 3.2. *The T-shape chunk sketch S_i returned by Algorithm 1 is in the form of a randomized quantile sketch consisting of H_c compactors, where $|S_i[H_c]| = \lfloor \frac{N_c}{2^{H_c-1}} \rfloor$, $|S_i[h]| = \lfloor \frac{N_c}{2^{h-1}} \rfloor \mod 2 \leq 1$ and $m_h = 1, 1 \leq h < H_c$.*

Proof. For the size of compactors, Lines 4 and 5 in the algorithm determine that $|S_i[h]| = \lfloor \frac{N_c}{2^{h-1}} \rfloor \mod 2$, while Line 8 brings $|S_i[H_c]| = \lfloor \frac{N_c}{2^{H_c-1}} \rfloor$. As for the number of compactions m_h, the h-th execution of Line 6 corresponds to a randomized compaction at height h: The case of the random bit being 0 is equivalent to keeping odd-indexed items in the compaction, and the case of being 1 is keeping even-indexed items. Thus, there is $m_h = 1, 1 \leq h < H_c$. □

Now we show that the constructed chunk sketch is optimal. That is, under the same size limit on sketch, it has the minimal probability of the estimation error exceeding ϵN_c for any ϵ.

Proposition 3.3. *The T-shape chunk sketch S_i built for chunk C_i is optimal, i.e., for any x and ϵ, there does not exist any other randomized sketch S_i' with the same size having*

$$\Pr[|R(x, S_i') - R(x)| > \epsilon N_c] \ < \ \Pr[|R(x, S_i) - R(x)| > \epsilon N_c].$$

Proof. Since the error of randomized sketches is zero-mean sub-Gaussian as introduced in Section 2.1, according to the Chernoff tail bound for sub-Gaussian variables, the proposition holds if the variance of error is shown to be minimal. Invoke Lemma 3.2, we have that the height $H_c = 1 + \lceil \log_2 \frac{N_c}{M_c} \rceil$ and number of compactions $m_h = 1$ of the proposed sketch reaches their minimal values for any N_c, M_c. Recall that the bound of variance of error is $\sigma^2 \leq \sum_{h=1}^{h=H_c-1} 4^{h-1} m_h$ as in Section 2.1, the proposed chunk sketch reaches the optimal bound and the proposition holds. □

There are conceptual differences between summarizing chunk data with the proposed T-shape sketch and sampling or KLL. Sampling selects samples from N_c data in the chunk, KLL treats N_c data as streaming input and summarizes them with multiple compactions, while the proposed T-shape sketch summarizes N_c in-memory data with the least compactions according to Lemma 3.2.

3.1.3 I/O Cost Analysis. For chunk sketches, their space on disk is equal to the I/O cost in the accelerated query process. Here we analyze the required I/O cost, i.e., the total size of pre-computed chunk sketches, to provide ϵN error guarantee on N queried data.

Proposition 3.4. *Given chunk size N_c, for any ϵ, δ, there exists randomized chunk sketches guaranteeing $\Pr\left[\left|\hat{R}(y) - R(y)\right| \leq \epsilon N\right] \geq 1 - \delta$ for any value y whose total size is $O\left(\frac{1}{\epsilon}(\log\frac{1}{\delta})^{\frac{1}{2}} N^{\frac{1}{2}} / N_c^{\frac{1}{2}} + N/N_c\right)$.*

Proof. Invoke Lemma 3.2 about m_h, the variance of error of N/N_c chunk sketches is bounded as $\sigma^2 \le (N/N_c)\sum_{h=1}^{H_c-1} m_h \cdot 4^{h-1} \le \frac{N}{3N_c} 4^{H_c-1}$. Apply the Chernoff bound for sub-Gaussian variables, the target error guarantee needs $\sigma^2 \le \frac{1}{2}\epsilon^2 N^2 / \log\frac{2}{\delta}$. It holds when $\frac{N}{3N_c} 4^{H_c-1} \le \frac{1}{2}\epsilon^2 N^2 / \log\frac{2}{\delta}$ and we get the upper bound of 2^{H_c} as $2^{H_c} \le \sqrt{6N_c N}\epsilon\sqrt{\log(\delta/2)}$. Note that the chunk sketch should be non-null, i.e., $2^{H_c} \le N_c$. Thus the required size of a chunk sketch is $O(N_c/2^{H_c-1}) = O(\frac{1}{\epsilon}\cdot\sqrt{\log\frac{1}{\delta}}\cdot\sqrt{N_c/N}+1)$. By summing up the size of N/N_c chunk sketches, we have the proposition. □

Intuitively, the proof shows that for the fixed ϵ, as more data are queried, the required size of each chunk sketch decreases at a speed of $\sqrt{N}$ but is at least $O(1)$. That means small chunk sketch size M_c can still ensure the ϵN error guarantee in queries when N is large. Compared with Proposition 3.3 about the error in individual chunk, Proposition 3.4 is the more important result on improving asymptotic I/O complexity of quantile queries with error guarantees.

However, this result is hard to be utilized in pre-computation since queries are unknown. For the same chunk size limit in pre-computation, the number of chunks is linear to N and so as the I/O cost. It motivates us to design sketches for SSTables since their size can grow as more data are ingested.

3.2 Hierarchical SSTable Sketches

Randomized sketches for SSTables are constructed during LSM-tree compaction events. Discussions in this section are based on LSM-tree with size-tiering merge policy. Due to the partitioning techniques in LSM-tree, not all data in the SSTable will be fetched in memory. Thus we construct SSTable sketches based on chunk sketches. In the following, we present Algorithm 2 of building sketches for an SSTable, the error analysis of SSTable sketches in Proposition 3.6, and the asymptotic I/O cost in Proposition 3.7.

3.2.1 Construction Algorithm. The two main ideas about SSTable sketches are: (1) Analyses about chunk sketches allow further compressing chunk sketches to improve the I/O complexity. (2) Building only one sketch for the whole SSTable has a low chance to utilize due to not all data are queried, data updates or overlapping key ranges in LSM-tree.

Sketches for a level L SSTable is organized as a tree with a branching factor T and a height at most L. For the further compression, we introduce a parameter of sketch size ratio T_s. A sketch at height h is at most T_s times larger than a sketch at height $h-1$, having $T_s < T$. The construction process shown in Algorithm 2 is like building a T-fork segment tree from bottom to top. The k chunk sketches in the SSTable are the leaves of the tree. The i-th sketch at height h is the result of combining and further compacting T sketches at height $h-1$. To be detailed, we combine T compactors at each height, and perform randomized compactions from bottom to top until the sketch size does not exceed the limit. The size limit is T_s times the average size of the combined T sketches.

Example 3.5. Figure 5 shows an example of building sketches for a level 2 SSTable with 4 chunks when $T = 2, T_s = 1$. All sketches generated have the same size since $T_s = 1$. When generating $S_{[1,4]}$, we combine $S_{[1,2]}$, $S_{[3,4]}$ and perform a randomized compaction at height 2 (keeping 1 in $\{1, 2\}$) and another at height 4 (keeping $\{5, 9\}$ in $\{4, 5, 7, 9\}$). Due to space limit, other possible results from randomized compactions are not shown.

3.2.2 Error Analysis. Since the top sketches take more randomized compactions (Line 5), they are less accurate (i.e., more error introduced) than the chunk sketches. We show below that the

Algorithm 2: Building sketches for an SSTable

Input: k chunk sketches $S_1, \ldots, S_k$, LSM-tree size ratio T, sketch size ratio T_s
Output: several sketches $S_{[l,r]}$ each representing a segment $[l, r]$ of chunks.

```
S_[i,i] ← S_i for 1 ≤ i ≤ k;
for h ← 1 to ⌈log_T k⌉ do
    len ← T^h; // Length of segment
    for i ← 1 to ⌊k/len⌋ do
        l ← 1 + (i − 1) · len; // Left endpoint
        S_[l,l+len−1] ←combine and further compact {S_[l+(j−1)·len/T, l+j·len/T−1] | 1 ≤ j ≤ T};
    end
end
```

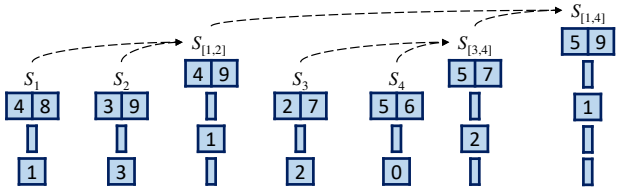


Fig. 5. Example of building hierarchical sketches for a level 2 SSTable with 4 chunks when $T = 2, T_s = 1$. Sketches are organized as an implicit segment tree. $S_{[1,4]}$ is built by combining and further compacting $S_{[1,2]}$ and $S_{[3,4]}$.

error guarantee provided by $S_{[1,T^L]}$ is still comparable to that of the combined chunk sketches $\{S_1, ..., S_{T^L}\}$.

Proposition 3.6. *Given $\sqrt{T}<T_s<T$, in a level L SSTable, if chunk sketches $\{S_1, ..., S_{T^L}\}$ guarantees $\Pr\left[\left|\hat{R}(y)-R(y)\right| \le \epsilon N\right] \ge 1-\delta$, the top sketch $S_{[1,T^L]}$ guarantees that $\Pr\left[\left|\hat{R}(y)-R(y)\right| \le C \cdot \epsilon N\right] \ge 1-\delta$ where $C = \sqrt{1+\frac{4^{H_c}}{T-(T/T_s)^2}}$, H_c is the height of chunk sketches.*

Proof. According to the Chernoff bound, the key idea is again to analyze and compare the bound $\sum 4^{h-1} m_h$ of variance of estimation error. For combined chunk sketches $\{S_1, ..., S_{T^L}\}$, there is $m'_h = T^L$, $1 \le h < H_c$, since numbers of compactions in a chunk sketch is known (Lemma 3.2). For $S_{[1,T^L]}$, its height is $H_c + L \cdot \log \frac{T}{T_s}$ and the number of compactions is

$$m_h = \begin{cases} T^L, & 1 \le h < H_c \\ T^{L-i-1}, & H_c + i \cdot \log \frac{T}{T_s} \le h < H_c + (i+1) \cdot \log \frac{T}{T_s}, 0 \le i < L \end{cases}.$$

Thus $\sum_{h=1}^{H_c+L\cdot\log\frac{T}{T_s}} 4^{h-1} m_h \le \left(1+\frac{4^{H_c}}{T-(T/T_s)^2}\right) \cdot \sum_{h=1}^{H_c} 4^{h-1} m'_h$ when $T_s > \sqrt{T}$. Invoking Chernoff bound concludes the proof. □

3.2.3 I/O Cost Analysis. The I/O cost to analyze is the total size of sketches utilized in the query to provide ϵN error guarantee on N queried data. Intuitively, the I/O cost is determined by data size N, precision ϵ, SSTable size ratio T in LSM-tree, SSTable sketches size ratio T_s, chunk size N_c and chunk sketch height H_c. Among the parameters, N_c and H_c determine chunk sketch size, while T and T_s influence SSTable sketch size. T, T_s and N_c are fixed in the LSM-tree. Since larger H_c brings

Table 2. Space complexity of disk-bounded solutions with different pre-computed sketches, when $T = 10, T_s = 5$.

	I/O & Memory
DeterChunk	$O\left(\frac{1}{\epsilon} \cdot N/N_c\right)$
RandChunk	$O\left(\frac{1}{\epsilon}(\log\frac{1}{\delta})^{\frac{1}{2}} N^{\frac{1}{2}}/{N_c}^{\frac{1}{2}} + N/N_c\right)$
RandSSTable	$O\left(\sqrt{\frac{1}{\epsilon}(\log\frac{1}{\delta})^{\frac{1}{2}}} N^{0.45}/{N_c}^{\frac{1}{4}} + (N/N_c)^{0.7}\right)$

smaller I/O but larger error as in the proof of Proposition 3.6, the idea for computing required I/O is to analyze the maximum H_c under ϵN error guarantee. The result is as follows.

Proposition 3.7. *Given N_c, $T > 0$, $T_s \in (\sqrt{T}, T)$, there exists randomized SSTable sketches guaranteeing* $\Pr\left[\left|\hat{R}(y) - R(y)\right| \le \epsilon N\right] \ge 1 - \delta$ *for any value y whose corresponding I/O cost during the query is* $O\left(\sqrt{\frac{1}{\epsilon}(\log\frac{1}{\delta})^{\frac{1}{2}}} N^{\left(-\frac{1}{4}+\log_T T_s\right)}/{N_c}^{\frac{1}{4}} + (N/N_c)^{\log_T T_s}\right)$.

Proof. Since there are at most T SSTables at the largest level of LSM-tree with size-tiering policy, the asymptotic I/O cost in query is the required size of the top sketch in the largest SSTable. Based on the proofs of Propositions 3.4 and 3.6, the variance of error in that top sketch is bounded with $\sigma^2 \le \left(1 + \frac{4^{H_c}}{T - (T/T_s)^2}\right) \frac{N}{3N_c} 4^{H_c - 1}$. Invoke the Chernoff bound, the target error guarantee needs $\sigma^2 \le \frac{1}{2}\epsilon^2 N^2 / \log\frac{2}{\delta}$. Thus we get the upper bound of 2^{H_c} as $16^{H_c} = O(\epsilon^2 N_c N \log\frac{2}{\delta})$. Similar to the proof of Proposition 3.4, chunk sketch is non-null and it brings another upper bound of 2^{H_c} as $2^{H_c} \le N_c$. Since the size of the top sketch grows T_s times when N grows T times, the I/O complexity is $O(N^{\log_T T_s}/2^{H_c})$. Apply the upper bound of 2^{H_c} and we have the proposition. □

Since the I/O complexity result in Proposition 3.7 is complicated with many factors, we describe each term to help understanding. The first part of the first term in the I/O complexity, $\sqrt{\frac{1}{\epsilon}(\log\frac{1}{\delta})^{\frac{1}{2}}}$, is influenced by the required error guarantee. That is, the ϵ closer to 0 needs larger sketches. The second part of the first term, $N^{\left(-\frac{1}{4}+\log_T T_s\right)}$, is influenced by the queried data size N. According to the proof above, we have that for ϵN error guarantee, the upper bound of 2^{H_c} is proportional to $N^{\frac{1}{4}}$. Thus the required size of each chunk sketch $N_c/2^{H_c}$ is proportional to $N^{-\frac{1}{4}}$. The last term $(N/N_c)^{\log_T T_s}$ comes from the limit that chunk sketch is non-null. It is the I/O cost when all chunk sketches are of size 1 and independent of ϵ about error guarantee.

To further illustrate the conclusion in simplified form and show the improved I/O complexity in real systems, we compare the results in Table 2 under a typical setting of $T = 10$ in LSM-trees, and the size ratio of SSTable sketches as $T_s = 5$. In Table 2, the DeterChunk is to keep the values with rank $\epsilon N_c, 2\epsilon N_c, \ldots$ deterministically as a summary of the chunk. The RandChunk and RandSSTable correspond to the proposed randomized sketches. All solutions simply keep queried pre-computed sketches in memory. That is, these disk-bounded solutions have a memory cost equal to the I/O cost during queries.

4 Memory-Constrained Solutions

The disk-bounded solutions so far have a memory cost equal to the I/O cost, which is not comparable with the memory bounds provided by streaming algorithms like the $O\left(\frac{1}{\epsilon}(\log\frac{1}{\delta})^{\frac{1}{2}}\right)$ of mergeable KLL [27]. The natural idea is to treat the pre-computed sketches as weighted input and apply the

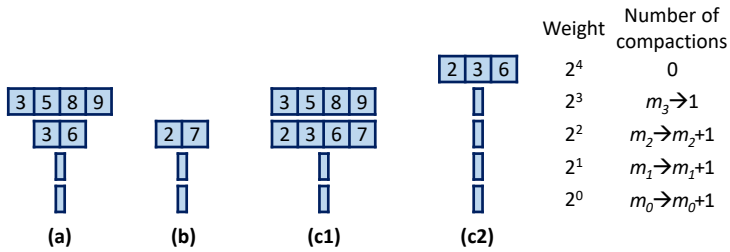


Fig. 6. Process of merging (a) the size-limited KLL sketch S in memory with (b) a pre-computed chunk sketch S_i, resulting in (c2) the merge result after compaction.

existing algorithms to compress them in memory. Here we apply the KLL algorithm, i.e., setting exponentially decreasing capacities for compactors maintained in memory. For these memory-constrained solutions, our contribution is to show that the improvement in I/O cost still exists while reaching the same memory bound as streaming algorithms.

4.1 Process of Merging Sketch

Algorithm 3 illustrates the process of merging a pre-computed randomized sketch S' on disk into the size-limited KLL sketch S in memory. As shown, we first combine compactors at same height by Merge Sort in Line 1, and then compact compactors exceeding their capacity from bottom to top as in Line 4.

Algorithm 3: Memory-constrained merging

Input: A KLL sketch S of H compactors with parameters K and γ, a pre-computed sketch S' of H' compactors to be merged
Output: The merged result S

```
1 for h ← 1 to H' do S[h] ← MergeSort(S[h], S_i[h]) ;
2 while true do
3 |   for h ← 1 to H − 1 do
4 |   |   if |S[h]| > K γ^{H−h} then compact S[h] ;
5 |   end
6 |   if |S[H]| ≤ K then return ;
7 |   compact S[H];
8 |   H ← H + 1;
9 end
```

Example 4.1. Figure 6 shows the process of merging a pre-computed chunk sketch S_i into the in-memory KLL sketch S with $K = 4$. As shown in (c1), we first combine compactors at same height and reorder them individually. Then we check the size of compactors from bottom to top. $S[3]$ is detected and compacted, making $S[4] = [2, 3, 5, 6, 8, 9]$ exceeds capacity K. The following compaction at height 4 increases the sketch height H to 5, and the final result is shown in (c2). Note that the increase of m_1, m_2 comes from accumulating the compactions in pre-computation of S_i.

4.2 I/O Cost Analysis

Now we analyze the required I/O cost of the memory-constrained solutions with pre-computed sketches, while keeping the memory bound $O\left(\frac{1}{\epsilon}(\log\frac{1}{\delta})^{\frac{1}{2}}\right)$ of the streaming mergeable KLL [12].

We start with analyzing when merging pre-computed chunk sketches is more accurate than streaming settings. The key is to compare the bound $\sum 4^{h-1}m_h$ of variance of error.

Proposition 4.2. *Given any capacity of the top compactor $K > 0$, capacity decrement factor $\gamma \in (\frac{1}{2}, 1)$ and chunk sketch height $H_c \geq 2$, if the KLL sketch S constructed from N streaming data guarantees* $\Pr\left[\left|\hat{R}(y)-R(y)\right| \leq \epsilon N\right] \geq 1-\delta$, *the KLL sketch S' summarizing pre-computed chunk sketches guarantees* $\Pr\left[\left|\hat{R}(y)-R(y)\right| \leq \epsilon' N\right] \geq 1-\delta$ *and $\epsilon' \leq \epsilon$ when $N \geq K^{1-\log_\gamma 2} \cdot N_c^{\log_\gamma 2} \cdot F(\gamma, H_c)$, where $F(\gamma, H_c)$ is a constant expression having*

$$F(\gamma, h) = \frac{1}{4-2\gamma} \cdot \left(\frac{3\gamma}{(2\gamma-1)(1-\gamma)}\right)^{\log_\gamma 2} \cdot \left(\frac{(2\gamma)^{h-1}-1}{4^{h-1}-1}\right)^{\log_\gamma 2}.$$

Proof. According to the Chernoff bound, the key idea is to analyze and compare the bound of variance of error in S' and S, i.e., $\sum 4^{h-1}m'_h$ and $\sum 4^{h-1}m_h$. We have $m_h = m'_h, h \geq H_c$ since the two methods only differ in the bottom $H_c - 1$ compactors. For $h \leq H_c - 1$, we have $m'_h = N/N_c$ from Lemma 3.2 and $m_h \geq 2N(\gamma/2)^h/(K\gamma^H)$ from [27]. Thus the merged S' is more accurate when $\sum 4^{h-1}N/N_c \leq \sum 4^{h-1}2N(\gamma/2)^h/(K\gamma^H)$ and then it is $K\gamma^H \leq \frac{3}{2}\frac{(2\gamma)^{H_c}-2\gamma}{(2\gamma-1)(4^{H_c-1}-1)}N_c$. That means H should be large enough as $\gamma \in (\frac{1}{2}, 1)$. Note that the sketch height H increases as N grows. Since a sketch with height $H-1$ summarizes at most $\sum_{h=1}^{h=H-1} 2^{h-1}K\gamma^{H-1-h}$ data, the lower bound of N given H is $N \geq \frac{K}{4-2\gamma}2^H$. Apply the requirement of H and we have the proposition. □

Intuitively, for the streaming sketch S, as N grows, the sketch height H grows, capacities $K\gamma^{H-h}$ decreases and compactions at some height h happen more and more frequently, i.e., m_h is super-linear to N. In contrast, for the merged sketch S', $m_h = N/N_c$ is linear to N. Thus a large enough N makes merging chunk sketches have smaller $\sum 4^{h-1}m_h$ and more accurate.

For example, when $\gamma = \frac{2}{3}$, $K = 2^{13}$, $N_c = 2^{13}$, $H_c = 5$, Proposition 4.2 states that for $N \geq 3.4 * 10^6$, summarizing chunk sketches is better than streaming settings. It is prevalent to query over 10^7 data in LSM-based systems handling intensive writes.

Now we can compute the required I/O cost of chunk sketches while providing a memory bound. The basic idea is that, for any KLL parameters K, γ, we find the requirement on 2^{H_c} about chunk sketches so that summarizing chunk sketches outperforms streaming settings, and then calculate the I/O cost of $O(N/2^{H_c})$.

Proposition 4.3. *For any $\epsilon > 0$, $\delta > 0$ and capacity decrement factor $\gamma \in (\frac{1}{2}, 1)$, there exists randomized chunk sketches and a KLL sketch summarizing them with $O\left(\frac{1}{\epsilon}(\log\frac{1}{\delta})^{\frac{1}{2}}\right)$ memory and $O\left((N/N_c)^{1/\left(1+\log\frac{1}{\gamma}\right)} + N/N_c\right)$ I/O complexity, that guarantees* $\Pr\left[\left|\hat{R}(y)-R(y)\right| \leq \epsilon N\right] \geq 1-\delta$.

Proof. According to the proof in Proposition 4.2, to make summarizing chunk sketches outperforms streaming, $K\gamma^H = O((\gamma/2)^{H_c}N_c)$ is required and thus $(2/\gamma)^{H_c} = O(\frac{N_c}{\gamma^H K})$ is sufficient. Then we get the upper bound of 2^{H_c} as $2^{H_c} = O((\frac{N_c}{\gamma^H K})^{1/\log(2/\gamma)})$. Note that $\gamma^H K$ is the capacity of the bottom compactor, which is $O(\gamma^{\log N})$. Then the I/O complexity is computed as $N/2^{H_c} = O\left(N/\left(N_c(N)^{\log(1/\gamma)}\right)^{1/\log(2/\gamma)}\right) = O\left(N^{1-\log\frac{1}{\gamma}/\log\frac{2}{\gamma}}/N_c^{1/\log\frac{2}{\gamma}}\right) = O\left((N/N_c)^{1/\left(1+\log\frac{1}{\gamma}\right)}\right)$, which is the

Table 3. Space complexity of memory-constrained solutions, when $\gamma = \frac{2}{3}, T = 10, T_s = 5$

	I/O	Memory
KLLStream[27]	$O(N)$	$O\left(\frac{1}{\epsilon}(\log\frac{1}{\delta})^{\frac{1}{2}}\right)$
RandChunk	$O\left(N^{0.63}/{N_c}^{0.63} + N/N_c\right)$	$O\left(\frac{1}{\epsilon}(\log\frac{1}{\delta})^{\frac{1}{2}}\right)$
RandSSTable	$O\left(N^{0.51}/{N_c}^{0.32} + (N/N_c)^{0.7}\right)$	$O\left(\frac{1}{\epsilon}(\log\frac{1}{\delta})^{\frac{1}{2}}\right)$

first term of the result. Again the second term of the I/O complexity comes from that chunk sketches are non-null. Since it is more accurate than streaming settings, the memory bound is also $O\left(\frac{1}{\epsilon}(\log\frac{1}{\delta})^{\frac{1}{2}}\right)$. □

Similarly, we analyze the I/O cost of compressing SSTable sketches with memory constraints.

Proposition 4.4. *For any $\epsilon > 0, \delta > 0, \gamma \in (\frac{1}{2}, 1), \sqrt{T} < T_s < T$, there exists randomized SSTable sketches and a KLL sketch summarizing them with a memory complexity of $O\left(\frac{1}{\epsilon}(\log\frac{1}{\delta})^{\frac{1}{2}}\right)$ and I/O complexity of $O\left(\frac{N^{\left(\frac{\log T_s}{\log T}(1-\frac{\log 2\gamma}{\log T\cdot\log\frac{8}{\gamma}})-\log\frac{1}{\gamma}/\log\frac{8}{\gamma}\right)}}{N_c^{\left(1-\frac{\log T_s}{\log T}\frac{\log 2\gamma}{\log T}\right)/\log\frac{8}{\gamma}}} + (\frac{N}{N_c})^{\log T_s/\log T}\right)$, that guarantees $\Pr\left[\left|\hat{R}(y)-R(y)\right| \le \epsilon N\right] \ge 1-\delta$.*

Proof. Since Proposition 3.6 bounds the variance of error in top SSTable sketch with that of chunk sketches, the key idea is to apply that bound and invoke the proof process in Proposition 4.3. Let the maximum level of SSTable to be L, the height of the top sketch in a level L SSTable is $H_c + L\log(T/T_s)$. Again, we analyze the bound of variance of error. According to Proposition 3.6 and the proof in Proposition 4.2, for compactors at height below $H_c + L\log(T/T_s)$, the variance of error of merged sketch S is smaller than that of streaming S' when $(1+\frac{4^{H_c}}{T-(T/T_s)^2})\frac{N}{N_c}\sum_{h=1}^{H_c-1}4^{h-1} \le \sum_{h=1}^{H_c+L\log(T/T_s)-1}4^{h-1}m'_h$. That holds when $\frac{4_c^H}{T-(T/T_s)^2}K\gamma^H = O\left((\frac{\gamma}{2})^{H_c}(2\gamma)^{L\log_T T_s}N_c\right)$. Note that L grows with N as $N_cT^L = N$ and thus there is $(2\gamma)^{L\log_T T_s} = (N/N_c)^{\log_T T_s\log_T 2\gamma}$. So the upper bound of 2^{H_c} is computed as $2^{H_c} \le \left(\frac{(T-(T/T_s)^2)N_c}{K\gamma^H}\right)^{1/\log(8/\gamma)}\left(\frac{N}{N_c}\right)^{\log_T T_s\log_T 2\gamma/\log(8/\gamma)}$. Another upper bound of $2^{H_c} \le N_c$ comes from non-null chunk sketches again. Apply the upper bounds of 2^{H_c} above and the I/O complexity $O(N^{\log_T T_s}/2^{H_c})$ is computed to be the proposition. More detailed process can be found in [36]. □

Since the I/O complexity above is complicated, we provide interpretations to help reading. The latter term of the I/O complexity is the same as that in Proposition 3.7, which comes from the limit of chunk sketch is non-null. The first term comes from that the required chunk sketch size decreases as N grows, similar to that in Proposition 3.7. Note that the I/O complexity is independent of ϵ, δ.

Table 3 summarize the space complexity for ϵN error guarantee in the common settings of $\gamma = \frac{2}{3}$ for KLL sketch in memory, $T = 10$ for LSM-tree, and $T_s = 5$ for SSTable sketches. Compared with disk-bounded solutions in Table 2, memory-constrained solutions need larger I/O cost for the better memory cost.

5 System Deployment

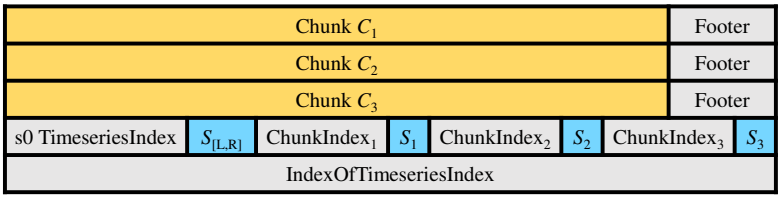


Fig. 7. Storing chunk sketches S_i and SSTable sketches $S_{[L,R]}$ in TsFile

The proposed solutions have been deployed in a time-series database Apache IoTDB [24, 37]. The system employs an LSM-tree based KV store [34] to handle intensive writes of time-series data in IoT scenarios, where the key of data is the generation timestamp and the value is the IoT sensor reading. The quantile query is on the value of data and the query statement is as follows.

```
SELECT quantile(series0, 0.5)
FROM root.sg0.device0
WHERE TIME >= 2022-01-01T00:00:00.000
AND   TIME <= 2022-01-03T00:00:00.000
```

The query is posed over a segment of time-series `series0` specified by the time range, and the result is the estimated 0.5-quantile of values. We present below more details of the deployment.

5.1 Sketch Construction and Storage

In Apache IoTDB, arrived data are first cached in memory, and flushed to disk when reaching the size limit. The basic unit of I/O is chunk (also named page). Chunk sketches are constructed by Algorithm 1 in Section 3.1 during the flush events. After an LSM-tree compaction event, SSTable sketches for the new SSTable are generated by Algorithm 2 in Section 3.2, but not all sketches must be rebuilt. For chunks with non-overlapped key ranges, the corresponding chunk sketches and SSTable sketches can be reused, since these chunk data are not changed in the compaction.

Figure 7 briefly illustrates the structure of TsFile, the format of SSTable in Apache IoTDB. The constructed chunk sketch S_i is stored together with the corresponding chunk index, thus we can read sketches without loading data in query processing. Similarly, the constructed hierarchical SSTable sketches, which height is larger than 1, are stored at the position of time-series index. We can read only SSTable sketches without loading chunk sketches.

5.2 Query with Sketches

The deployed query processor avoids visiting the data if the corresponding sketches can be utilized. When utilizing hierarchical sketches in the SSTable for a query with time range, the range will be decomposed as discussed in Section 3.2.3. The quantile queries are on the values of the KV-store, so sketches can be utilized if the key ranges of summarized files or chunks do not overlap. In IoTDB, key ranges of files and chunks are available for the query processor as meta data. Whether to use a sketch is decided by checking whether the corresponding key range is overlapped with others. There are overlapping sorted runs in the LSM-tree because keys of arrival data are unordered or there are updates. For time-series data, data loads with transmission delays [26] can cause unordered key arrivals. Overlapping runs limit the benefit of using sketches but do not impact correctness.

Figure 8 shows an example of querying on two overlapping SSTables. As in Figure 8(a), both SSTable consist of four chunks. SSTable 2 has the more recent version, thus the entry with timestamp 9 and value 4 is a update. The timestamp of time-series here serves as the key of KV data in Figure 1. In Figure 8(b)(c), the chunk sketch S_i summarizing chunk C_i is constructed from C_i

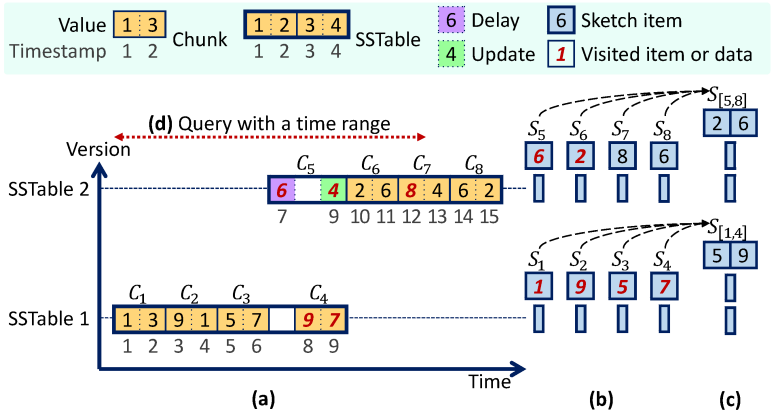


Fig. 8. Solution deployed in Apache IoTDB, with (a) key-range overlapping SSTables caused by delay or update, (b) chunk sketches, (c) SSTable sketches and (d) query posed over data in a time range

with Algorithm 1, and SSTable sketch $S_{[l,r]}$ summarizing chunks $\{C_l, ..., C_r\}$ is constructed from $\{S_l, ..., S_r\}$ with Algorithm 2. As in Figure 8(d), the query is posed on data specified by the time range $[1, 12]$. In the accelerated query processing, data with timestamps 7,8,9,12 are visited because of the overlapping or queried time range. Chunk sketches S_1, S_2, S_3, S_6 are utilized because the chunks are contained in the query and not overlapped.

5.3 Implementation in Other Systems

While our current implementation is on Apache IoTDB, it is highly possible to implement in other LSM-based systems like RocksDB. RocksDB supports the size-tiered compaction policy, mentioned Section 2.2. The chunk in Section 3.1 corresponds to the data block in RocksDB, while the SSTable corresponds to the file formatted as BlockBasedTable.

The chunk sketch in Section 3.1 summarizes values in a data block, and the SSTable sketches in Section 3.2 summarize the whole file. Based on the file format of RocksDB [17], to store the pre-computed sketches, new meta blocks need to be added in the file of RocksDB. One new meta block consists of chunk (data block) sketches and is maintained in the same way of the index block. Another new meta block consists of the hierarchical SSTable sketches and is maintained in the same way of the filter block [17].

Sketches summarize values of data in the KV-store, while they can be utilized in queries if the key ranges of summarized files or data blocks do not overlap. To check this in the query process, the key range of the file needs to be added into the stats block [17].

6 Experimental Evaluation

In experiments, we evaluate the performance of pre-computed sketches to verify propositions. The related code and data are available for reproducibility [5].

6.1 Experimental Setup

We implement solutions in Apache IoTDB [37]. The experiments are conducted on a machine with Intel(R) Core(TM) i7-11800H CPU @2.30GHz, 64GB DDR4 DRAM memory and 2TB NVMe SSD.

6.1.1 Datasets. We employ synthetic and real-world datasets with various distributions shown in Figure 9.

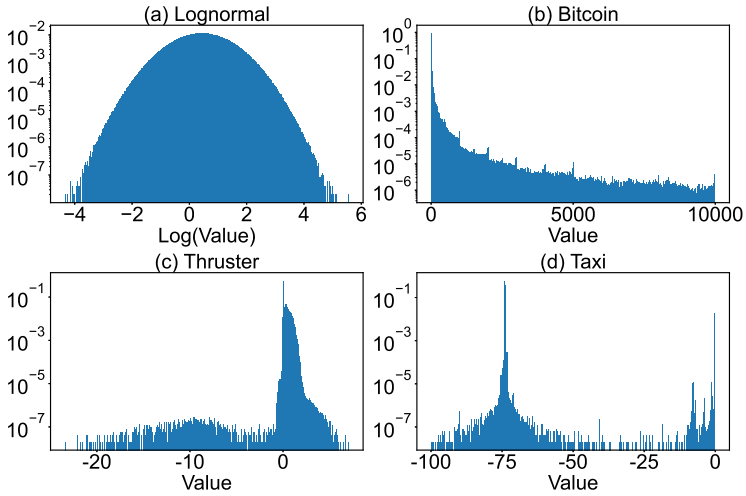


Fig. 9. Value distributions of employed datasets

(1) Lognormal is a synthetic dataset with 1×10^9 i.i.d. values sampled from a log-normal distribution [15] with a scale parameter of 1 and a shape parameter of 2.
(2) Bitcoin [9] is a public dataset on Kaggle, which records the transaction of Bitcoin since 2009.
(3) Thruster [11] is a public dataset on Kaggle, which is based on the physics of monopropellant chemical thrusters.
(4) Taxi [10] is a public dataset on Kaggle, recording longitude data about pick ups and drop offs of taxes.

6.1.2 Metrics. To measure the performance, we perform queries on different parts of data and report the average time cost, required I/O and actual error rate of estimated quantiles. The queried quantiles are $\{0.01\%, 0.02\%, ..., 99.99\%\}$.

6.1.3 Baselines. Besides solutions utilizing the proposed pre-computed randomized sketches, we implement several methods based on existing techniques as follows: (1) NoSketch loads all queried data in memory and finds the exact quantile with QuickSelect [20] algorithm. (2) DDSketch summarizes streaming data with a size-limited DDSketch [32]. (3) TDigest summarizes streaming data with a size-limited t-digest with scale function k_0 [16]. (4) DeterChunk keeps values with rank $\epsilon N_c, 2\epsilon N_c, ...$ deterministically as the chunk sketch. In the memory-constrained settings, DeterChunk takes the chunk sketches as weighted input to the upstream randomized compaction method KLL. (5) KLL summarizes streaming data with a KLL sketch [27] in memory.

6.1.4 Parameters. For LSM-tree, the parameters are a chunk size of 8KB ($N_c = 1024$), a Memtable size of 800KB and a component size ratio of $T = 10$. These parameters are close to the default settings in practical systems like LevelDB [28] and IoTDB [24]. For SSTable sketches, the size ratio is $T_s = 5$, consistent with the settings in Tables 2 and 3. For KLL sketch in memory applied in memory-constrained solutions, we set $\gamma = \frac{2}{3}$ as in the algorithms library Apache DataSketches [12].

6.2 Performance of Disk-Bounded Solutions

We first show the performance of disk-bounded solutions to verify Table 2 about I/O complexity. In this experiment the requirement on error guarantee is set to be an average error of 0.005. Figure

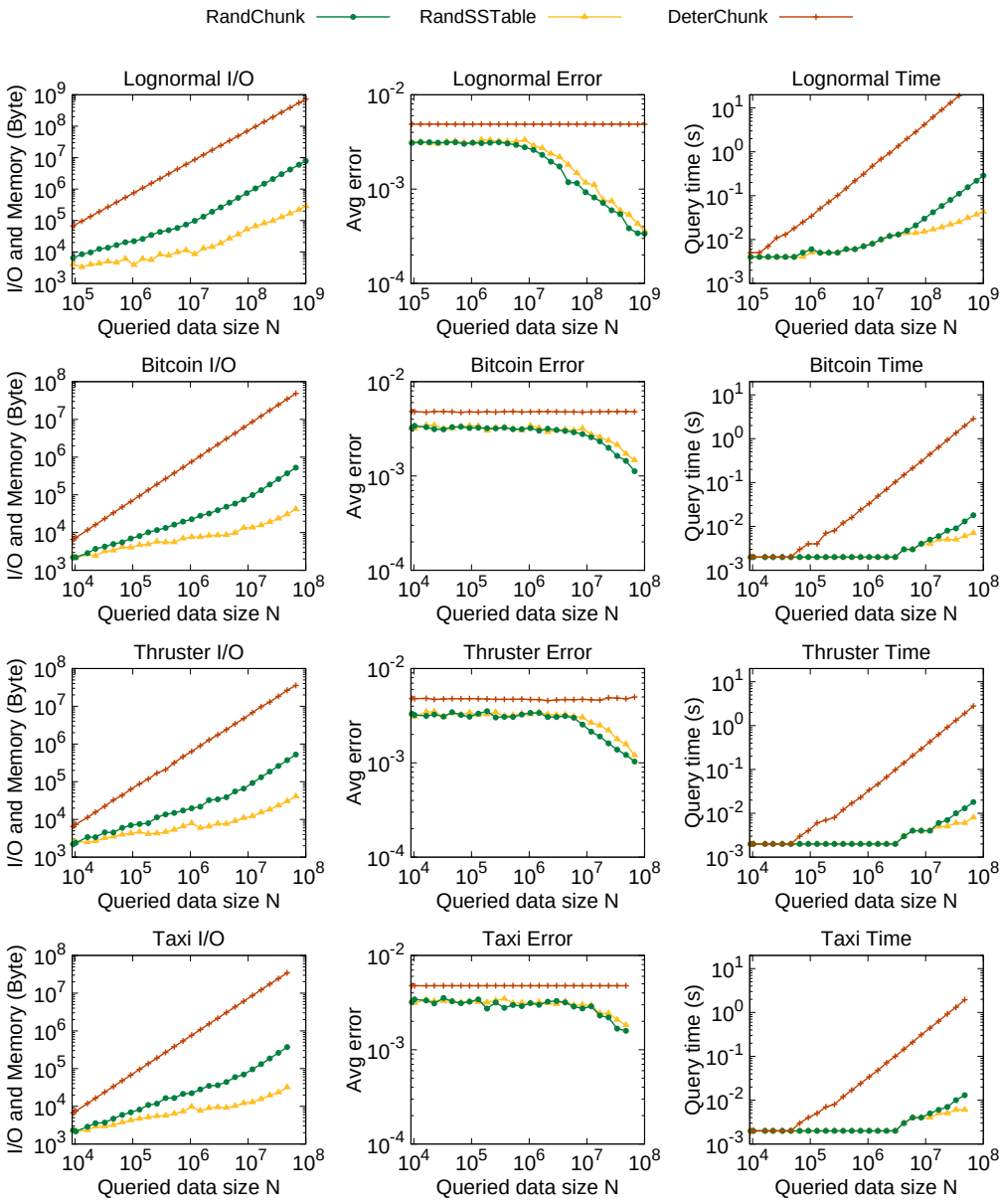


Fig. 10. Performance of disk-bounded solutions with varied N, to provide a guarantee of 0.005 average error

10 reports the required I/O cost as well as the corresponding actual error and query time cost with various data size N to achieve the required accuracy. Here the memory cost in query is not constrained and equal to the I/O cost.

For the I/O cost, the cost of DeterChunk is linear to N, i.e., the required size of each chunk sketch keeps the same. That is consistent with the $O\left(\frac{1}{\epsilon} \cdot N/N_c\right)$ in Table 2. The curve of randomized chunk

sketches (RandChunk) is proportional to $\sqrt{N}$ for small N, but becomes linear to N when $N \geq 1 \times 10^7$. That is because the required size of each randomized chunk sketch decreases as N grows and becomes 1 for large enough N. The curve of randomized SSTable sketches (RandSSTable) has a turning point for the same reason. Its trend line is proportional to $N^{0.25}$ and then $N^{0.71}$. Overall, the asymptotic behaviors are similar in different datasets and consistent with the upper bounds in Table 2. For example, the performance of RandSSTable for small N ($\propto N^{0.24}$ in subfigure (b1)) is better than the loose upper bound of $\propto N^{0.45}$ in the table.

For the average actual error, all results are below the required 0.005. When N is large enough and the size of chunk sketches becomes 1, the error of randomized sketches further decreases. In this case, the total size of sketches is determined by the non-null chunk sketch limit and larger than required by accuracy. For the query time cost, the performance is similar to that of the I/O cost. Unpacking the large SSTable on disk dominates the time cost when N is small. Thus RandSSTable significantly outperforms RandChunk when N is large enough.

6.3 Performance of Memory-Constrained Solutions

Now we compare the performance of solutions under the memory-constrained settings.

6.3.1 Varying N. Figure 11 shows the required I/O cost as well as the corresponding actual error and query time cost with various N to verify Table 3. Again, the required error guarantee is set to be an average error of 0.005 if possible. According to the space bound of streaming algorithm KLL, the memory limit is computed to be 2,644 bytes and applied to methods except NoSketch.

The NoSketch method loading the whole queried data occupies memory proportional to N and suffers from OutOfMemory error when N=1×10^9. NoSketch is significantly slower (by order-of-magnitude) than the proposed solution using randomized sketches. DDSketch has a time cost very close to KLL, while t-digest is much slower, which is consistent to results in [8]. Both DDSketch and TDigest summarize all queried data online and show large average rank error, as they do not provide rank error guarantees. DeterChunk takes the same I/O as in Figure 10 but does not achieve the error guarantee due to the memory constraint. In terms of I/O and query time, DeterSketch outperforms NoSketch but performs worse than the proposed solutions.

For RandChunk and RandSSTable, the turning point of I/O cost exists for the same reason of chunk sketch size reaching 1. The curve of RandChunk turns from $\propto N^{0.53}$ to $\propto N$. The curve of RandSSTable is proportional to $N^{0.22}$ and then $N^{0.70}$. In summary, results on all datasets are below the upper bounds in Table 3. The actual error of RandChunk and RandSSTable do not further decrease significantly after N is large enough, which is different from that in disk-bounded solutions. It is because the memory budget is limited in this experiment. The time cost of RandSSTable significantly outperforms RandChunk when N is large and the difference in I/O is large enough.

6.3.2 Varying Memory Constraint. Now we vary the memory constraint, referring to Algorithm 3 in Section 4 about query process. Similar results on some datasets are omitted. As shown in Figure 12, there is little change in time cost of all memory-constrained methods. The solution with SSTable sketches keeps two order of magnitude improvement in query time compared to the streaming methods (KLL, TDigest and DDSketch). In terms of error, all methods have better accuracy as the constrained memory grows. As the constrained memory grows, the error of solutions with pre-computation (RandChunk, RandSSTable and DeterChunk) become closer to that in disk-bounded experiments in Figure 10. Proposed solutions with pre-computed randomized sketches provide accuracy comparable to or better than baselines in all cases.

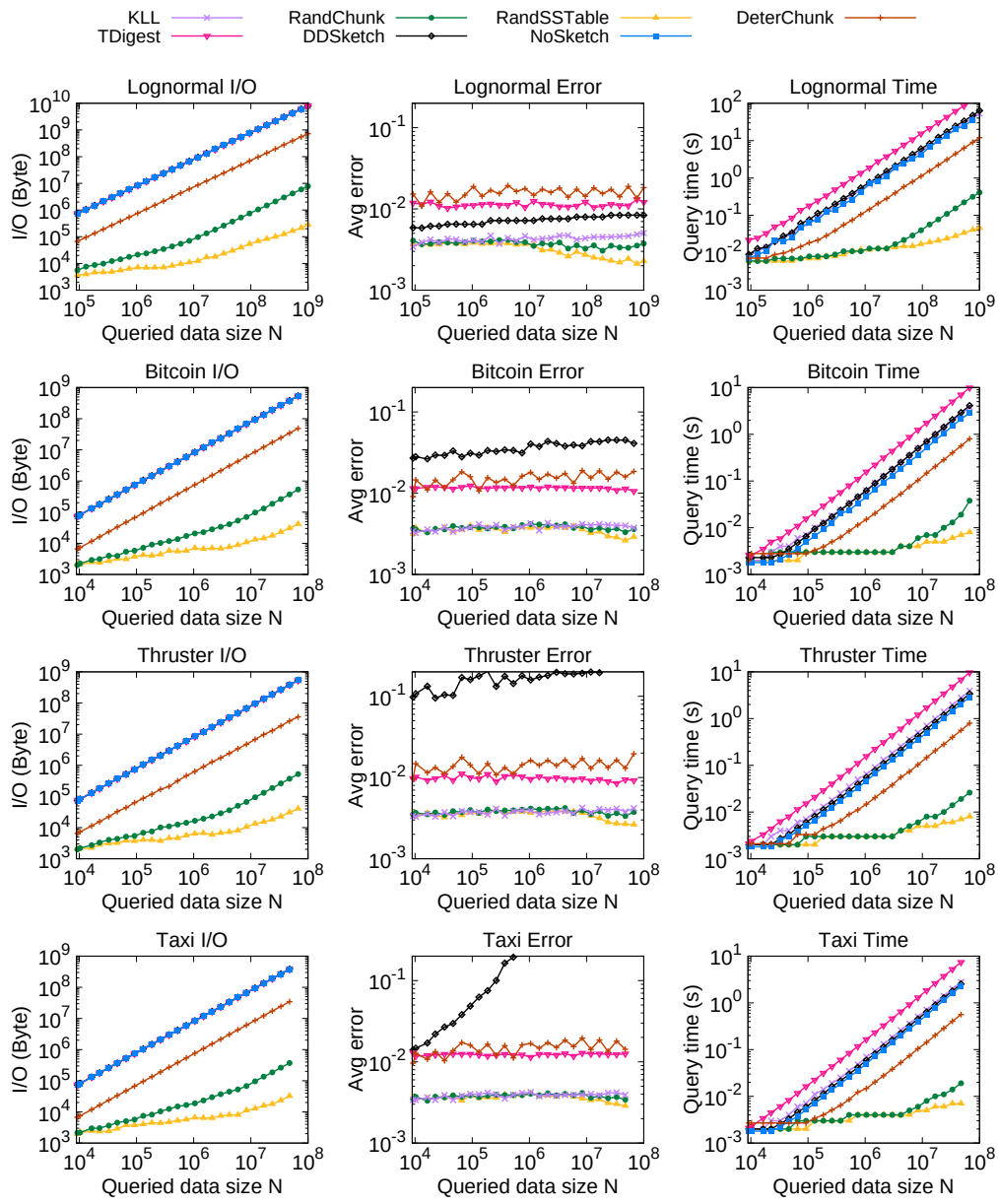


Fig. 11. Performance of memory-constrained solutions with varied N, to provide a guarantee of 0.005 average error if possible, within 2,664 bytes memory except the no-sketch solution

6.4 Performance of System Deployment

Finally, we evaluate the overall performance of Apache IoTDB with pre-computed sketches deployed.

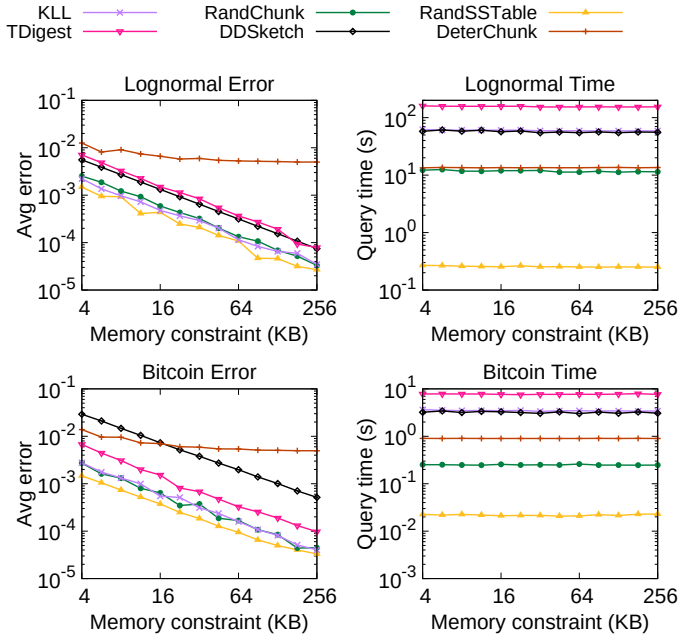


Fig. 12. Varying memory constraint

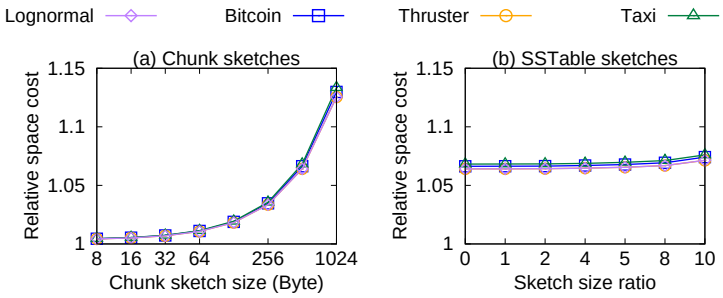


Fig. 13. Space amplification by pre-computed sketches, under (a) various chunk sketch size M_c and (b) various SSTable sketch size ratio T_s, relative to no pre-computation

6.4.1 Space Amplification Introduced by Sketches. Here we show the space amplification introduced by the sketches under various parameters in Figure 13. The space cost relative to no pre-computation grows as chunk sketch size M_c and SSTable sketch size ratio T_s. In Figure 13(b), M_c=512 bytes is fixed and the result shows that the SSTable sketches are much smaller than chunk sketches. Recall that in Sections 6.2 and 6.3, the chunk sketch size M_c can be close to 1 to achieve 0.005 average error under large N, which means a minimal impact on space amplification.

6.4.2 Write Performance Affected by Sketches. We show the effect of generating proposed sketches in a write-intensive scenario. Figure 14 shows the write throughput as time, with/without compaction, having M_c=512 bytes and T_s=5. The x-axis indicates the passage of time as the system

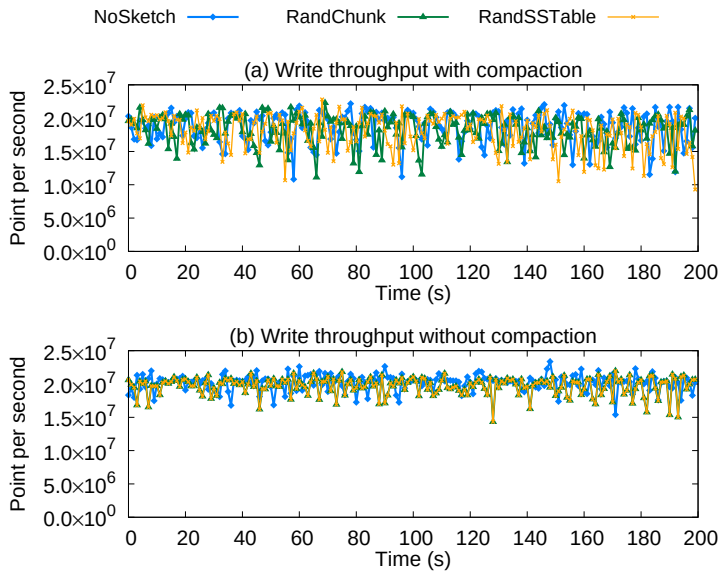


Fig. 14. Write throughput as time, (a) with compactions and (b) without compactions in the scenario

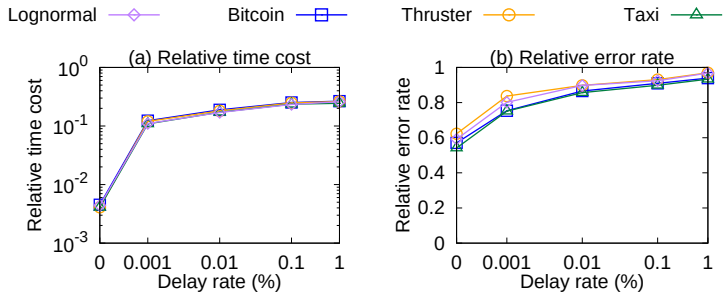


Fig. 15. Query performance improved by pre-computed sketches over the time-series data with various delay rates, relative to no pre-computation

operates, and the y-axis represents the write throughput. In Figure 14(a), compactions are triggered at some timestamps to limit the number of files, and considerably hinder the throughput, since it is not able to keep up with the insertion rate. Generating sketches further reduce the average throughput by about 3% compared with NoSketch, which is marginal. The reason is that generating sketches during compactions does not need additional scanning. In Figure 14(b), there is no compaction, the number of files grows as time and the average throughput is 8.3% higher than that in (a). Again, generating sketches reduce the average throughput by about 3%.

6.4.3 Query Performance in Out-of-order Data. As discussed in Section 5, data with out-of-order arrivals will impede query optimization with pre-computed summaries. We consider data loads with transmission delays based on a real-world case [26], and vary the rate of data suffered from severe delays.

Figure 15 shows the relative query performance of memory-constrained solution with SSTable sketches, compared to no pre-computation. As more data suffer from severe delays, the time cost

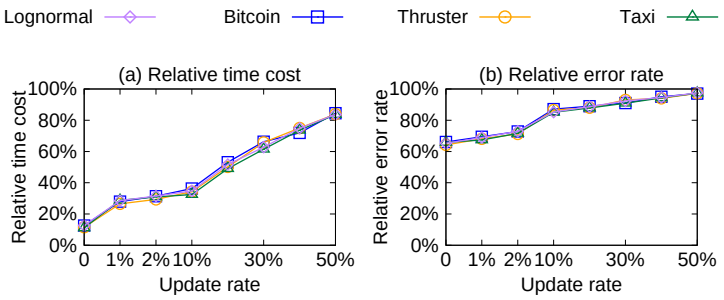


Fig. 16. Query performance improved by pre-computed sketches over the data with various update rates, relative to no pre-computation

and error rate of solution with pre-computations become closer to those of the streaming KLL, i.e., relative performance closer to 1, since fewer pre-computed summaries can be utilized. In real-world scenarios, most delays are minimal. The rates of severe delays observed from the case in [38] and the industrial use case in [26] are only 0.05% and 0.0375% respectively, where the improvements on time cost and error rate by pre-computed sketches are significant.

6.4.4 Query Performance in Updated Data. Due to the out-of-place update nature of LSM-tree, data updates will impede utilizing pre-computed summaries in query processing. Here we generate data loads such that each insert has a probability up to 50% to update a recent data point. Figure 16 shows the relative query performance of memory-constrained solution with SSTable sketches compared to no pre-computation. With more updates, more pre-computed summaries are no longer valid, and thus the performance becomes closer to that without pre-computation. Actually, any pre-computation including our proposed sketch is feasible for low-update scenarios such as IoT. The reason is that the pre-computation are no longer valid when the corresponding data are updated.

7 Related Work

The related studies are mainly in quantile algorithms and data summaries in LSM-tree based storage.

7.1 Quantile Estimation

The quantile problem in limited memory is first studied in [33]. The idea of sketching, i.e., leveled compactors, is proposed and then developed in the following decades. Sketches like MRL sketch [31] and GK sketch [18] are introduced. Agarwal et al. [2] propose the randomized compaction in the compactor. The KLL sketch [27] integrates these techniques and is proved to be optimal in space. Mergeability of quantile summaries is discussed in [2]. Algorithms library in Apache DataSketches [12] provides an implementation of fully mergeable KLL sketch.

In addition, other algorithms like t-digest [16] and DDSketch [32] are popular in practice. The t-digest adopts averaging and linear interpolation to summarize data, which can be attacked [8] but performs well in simple scenarios. DDSketch takes logarithm operation and performs well in estimating extreme quantiles of long-tailed distributions. These algorithms can not provide any rank error guarantees but are practical in some scenarios and thus covered as baselines in our study.

Sketches like KLL± [39] support data deletion. In other words, the pre-computed sketch can be updated and utilized in query. However, to determine the value to delete or update, the entire

chunk should be read from disk and decompressed. Hence, the deletion scheme of sketches does not improve the I/O cost and thus is not considered in this study.

Most recently, the ReqSketch [7] achieves the best known asymptotic behavior in providing the type of relative error guarantees. That is, more accurate estimations for more extreme quantiles. It is not considered in our study because in the problem of pre-computation, the extreme quantile on the whole dataset may be non-extreme quantiles on SSTables. Thus the relative error guarantee is not suitable for pre-computation.

7.2 Data Summaries in LSM-tree based Systems

LSM-tree based stores with immutable SSTables like Apache IoTDB [37] often pre-compute and store some simple and mergeable info such as the count of the data, to accelerate simple aggregation queries. For more complex data summaries, their mergeability must be considered. Absalyamov et al. [1] propose generating histograms and wavelets for indexed attributes to estimate cardinality in query with value range filter. The work does not actually merge the summaries since equi-height histograms are hard to merge. Singh et al. [35] combine disk-resident indexes and streaming algorithms to estimate quantiles. It needs to reorder disk data by their values, which is not practical in LSM-tree based store.

8 Conclusions

In this study, our major technical contributions are the pre-computation of randomized sketches in LSM-based store and the improved I/O complexity to provide error guarantees in quantile queries. First, we construct the chunk sketch over all the chunk data in memory, which is proved to be the randomized sketch with optimal error guarantees (Proposition 3.3). To achieve ϵN error guarantee, the required chunk sketch size decreases as N grows and brings an I/O cost proportional to $\sqrt{N}$ (Proposition 3.4). Next, we construct hierarchical sketches for each SSTable by compressing the chunk sketches, thus the asymptotic I/O cost is further improved (Proposition 3.7). Finally, we study the improvement in I/O while keeping a memory bound as that in streaming solutions, by compressing pre-computations with KLL algorithm. Analyses show that the in-memory KLL sketch summarizing pre-computations can be more accurate than summarizing streaming data (Proposition 4.2). In this memory-constrained setting, pre-computations still bring asymptotic improvement in I/O (Propositions 4.3 and 4.4). Tables 2 and 3 summarize our major theoretical contributions in common LSM-tree settings. The pre-computations are deployed in an LSM-tree based time-series database Apache IoTDB. Extensive experiments support the asymptotic behavior analyses and show the superiority of our proposals.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China (62232005, 62021002, 92267203, 62072265), the National Key Research and Development Plan (2021YFB3300500), Beijing National Research Center for Information Science and Technology (BNR2025RC01011), and Beijing Key Laboratory of Industrial Big Data System and Application. Shaoxu Song (https://sxsong.github.io/) is the corresponding author.

References

[1] Ildar Absalyamov, Michael J. Carey, and Vassilis J. Tsotras. 2018. Lightweight Cardinality Estimation in LSM-based Systems. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). ACM, 841–855. https://doi.org/10.1145/3183713.3183761

[2] Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff M. Phillips, Zhewei Wei, and Ke Yi. 2013. Mergeable summaries. *ACM Trans. Database Syst.* 38, 4 (2013), 26. https://doi.org/10.1145/2500128

[3] Burton H. Bloom. 1970. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM* 13, 7 (1970), 422–426. https://doi.org/10.1145/362686.362692

[4] Zhiwei Chen, Shaoxu Song, Ziheng Wei, Jingyun Fang, and Jiang Long. 2021. Approximating Median Absolute Deviation with Bounded Error. *Proc. VLDB Endow.* 14, 11 (2021), 2114–2126. https://doi.org/10.14778/3476249.3476266

[5] Experiment Code and Data. 2024. https://github.com/thssdb/LSM-quantile/.

[6] Graham Cormode. 2021. Current Trends in Data Summaries. *SIGMOD Rec.* 50, 4 (2021), 6–15. https://doi.org/10.1145/3516431.3516433

[7] Graham Cormode, Zohar S. Karnin, Edo Liberty, Justin Thaler, and Pavel Veselý. 2021. Relative Error Streaming Quantiles. In *PODS'21: Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Virtual Event, China, June 20-25, 2021*, Leonid Libkin, Reinhard Pichler, and Paolo Guagliardo (Eds.). ACM, 96–108. https://doi.org/10.1145/3452021.3458323

[8] Graham Cormode, Abhinav Mishra, Joseph Ross, and Pavel Veselý. 2021. Theory meets Practice at the Median: A Worst Case Comparison of Relative Error Quantile Algorithms. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, Feida Zhu, Beng Chin Ooi, and Chunyan Miao (Eds.). ACM, 2722–2731. https://doi.org/10.1145/3447548.3467152

[9] Bitcoin Dataset. 2018. https://www.kaggle.com/shiheyingzhe/bitcoin-transaction-data-from-2009-to-2018.

[10] Taxi Dataset. 2018. https://www.kaggle.com/datasets/sunnets/taxipredictiondata8m.

[11] Thruster Dataset. 2022. https://www.kaggle.com/datasets/patrickfleith/spacecraft-thruster-firing-tests-dataset.

[12] Apache Datasketches. 2024. https://datasketches.apache.org/.

[13] Niv Dayan, Manos Athanassoulis, and Stratos Idreos. 2017. Monkey: Optimal Navigable Key-Value Store. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu (Eds.). ACM, 79–94. https://doi.org/10.1145/3035918.3064054

[14] Niv Dayan and Stratos Idreos. 2018. Dostoevsky: Better Space-Time Trade-Offs for LSM-Tree Based Key-Value Stores via Adaptive Removal of Superfluous Merging. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). ACM, 505–520. https://doi.org/10.1145/3183713.3196927

[15] Lognormal Distribution. 2024. https://commons.apache.org/proper/commons-math/javadocs/api-3.6.1/org/apache/commons/math3/distribution/LogNormalDistribution.html.

[16] Ted Dunning. 2021. The t-digest: Efficient estimates of distributions. *Softw. Impacts* 7 (2021), 100049. https://doi.org/10.1016/j.simpa.2020.100049

[17] RocksDB File Format. 2024. https://github.com/facebook/rocksdb/wiki/Rocksdb-BlockBasedTable-Format.

[18] Michael Greenwald and Sanjeev Khanna. 2001. Space-Efficient Online Computation of Quantile Summaries. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*, Sharad Mehrotra and Timos K. Sellis (Eds.). ACM, 58–66. https://doi.org/10.1145/375663.375670

[19] Joseph M. Hellerstein, Christopher Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, and Arun Kumar. 2012. The MADlib Analytics Library or MAD Skills, the SQL. *Proc. VLDB Endow.* 5, 12 (2012), 1700–1711. https://doi.org/10.14778/2367502.2367510

[20] C. A. R. Hoare. 1962. Quicksort. *Comput. J.* 5, 1 (1962), 10–15. https://doi.org/10.1093/comjnl/5.1.10

[21] Document in Apache IoTDB. 2024. https://iotdb.apache.org/UserGuide/Master/Operators-Functions/Data-Profiling.html#quantile.

[22] Implementation in Apache IoTDB. 2024. https://github.com/apache/iotdb/tree/research/LSM-quantile.

[23] Quantile Query in Flux. 2024. https://docs.influxdata.com/influxdb/v2/query-data/flux/percentile-quantile/#estimate_tdigest.

[24] Apache IoTDB. 2024. https://iotdb.apache.org/.

[25] Christopher M. Jermaine, Edward Omiecinski, and Wai Gen Yee. 2007. The partitioned exponential file for database storage management. *VLDB J.* 16, 4 (2007), 417–437. https://doi.org/10.1007/s00778-005-0171-7

[26] Yuyuan Kang, Xiangdong Huang, Shaoxu Song, Lingzhe Zhang, Jialin Qiao, Chen Wang, Jianmin Wang, and Julian Feinauer. 2022. Separation or Not: On Handing Out-of-Order Time-Series Data in Leveled LSM-Tree. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 3340–3352. https://doi.org/10.1109/ICDE53745.2022.00315

[27] Zohar S. Karnin, Kevin J. Lang, and Edo Liberty. 2016. Optimal Quantile Approximation in Streams. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, Irit Dinur (Ed.). IEEE Computer Society, 71–78. https://doi.org/10.1109/FOCS.2016.17

[28] Google LevelDB. 2024. https://github.com/google/leveldb.

[29] Pengju Liu, Cui-Ping Li, and Hong Chen. 2024. Enhancing Storage Efficiency and Performance: A Survey of Data Partitioning Techniques. *J. Comput. Sci. Technol.* 39, 2 (2024), 346–368. https://doi.org/10.1007/S11390-024-3538-1

[30] Chen Luo and Michael J. Carey. 2020. LSM-based storage techniques: a survey. *VLDB J.* 29, 1 (2020), 393–418. https://doi.org/10.1007/s00778-019-00555-y

[31] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. 1999. Random Sampling Techniques for Space Efficient Online Computation of Order Statistics of Large Datasets. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh (Eds.). ACM Press, 251–262. https://doi.org/10.1145/304182.304204

[32] Charles Masson, Jee E. Rim, and Homin K. Lee. 2019. DDSketch: A Fast and Fully-Mergeable Quantile Sketch with Relative-Error Guarantees. *Proc. VLDB Endow.* 12, 12 (2019), 2195–2205. https://doi.org/10.14778/3352063.3352135

[33] J. Ian Munro and Mike Paterson. 1980. Selection and Sorting with Limited Storage. *Theor. Comput. Sci.* 12 (1980), 315–323. https://doi.org/10.1016/0304-3975(80)90061-4

[34] Patrick E. O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth J. O'Neil. 1996. The Log-Structured Merge-Tree (LSM-Tree). *Acta Informatica* 33, 4 (1996), 351–385. https://doi.org/10.1007/s002360050048

[35] Sneha Aman Singh, Divesh Srivastava, and Srikanta Tirthapura. 2016. Estimating Quantiles from the Union of Historical and Streaming Data. *Proc. VLDB Endow.* 10, 4 (2016), 433–444. https://doi.org/10.14778/3025111.3025124

[36] Supplementary. 2024. urlhttps://github.com/thssdb/LSM-quantile/blob/main/full.pdf.

[37] Chen Wang, Jialin Qiao, Xiangdong Huang, Shaoxu Song, Haonan Hou, Tian Jiang, Lei Rui, Jianmin Wang, and Jiaguang Sun. 2023. Apache IoTDB: A Time Series Database for IoT Applications. *Proc. ACM Manag. Data* 1, 2 (2023), 195:1–195:27. https://doi.org/10.1145/3589775

[38] Wolfgang Weiss, Víctor Juan Expósito Jiménez, and Herwig Zeiner. 2017. A Dataset and a Comparison of Out-of-Order Event Compensation Algorithms. In *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security, IoTBDS 2017, Porto, Portugal, April 24-26, 2017*, Muthu Ramachandran, Víctor Méndez Muñoz, Verena Kantere, Gary B. Wills, Robert John Walters, and Victor Chang (Eds.). SciTePress, 36–46. https://doi.org/10.5220/0006235400360046

[39] Fuheng Zhao, Sujaya Maiyya, Ryan Weiner, Divy Agrawal, and Amr El Abbadi. 2021. KLL±: Approximate Quantile Sketches over Dynamic Datasets. *Proc. VLDB Endow.* 14, 7 (2021), 1215–1227. https://doi.org/10.14778/3450980.3450990

[40] Xuanhe Zhou, Zhaoyan Sun, and Guoliang Li. 2024. DB-GPT: Large Language Model Meets Database. *Data Sci. Eng.* 9, 1 (2024), 102–111. https://doi.org/10.1007/S41019-023-00235-6

Received July 2024; revised September 2024; accepted November 2024