Beyond Simple Aggregates: Indexing for Summary Queries

Zhewei Wei Ke Yi
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong, China
{wzxac, yike}@cse.ust.hk

ABSTRACT

Database queries can be broadly classified into two categories: reporting queries and aggregation queries. The former retrieves a collection of records from the database that match the query's conditions, while the latter returns an aggregate, such as count, sum, average, or max (min), of a particular attribute of these records. Aggregation queries are especially useful in business intelligence and data analysis applications where users are interested not in the actual records, but some statistics of them. They can also be executed much more efficiently than reporting queries, by embedding properly precomputed aggregates into an index.

However, reporting and aggregation queries provide only two extremes for exploring the data. Data analysts often need more insight into the data distribution than what those simple aggregates provide, and yet certainly do not want the sheer volume of data returned by reporting queries. In this paper, we design indexing techniques that allow for extracting a statistical summary of all the records in the query. The summaries we support include frequent items, quantiles, various sketches, and wavelets, all of which are of central importance in massive data analysis. Our indexes require linear space and extract a summary with the optimal or near-optimal query cost.

Categories and Subject Descriptors

E.1 [**Data**]: Data structures; F.2.2 [**Analysis of algorithms and problem complexity**]: Nonnumerical algorithms and problems

General Terms

Algorithms, theory

Keywords

Indexing, summary queries

1. INTRODUCTION

A database system's primary function is to answer users' queries. These queries can be broadly classified into two categories: reporting queries and aggregation queries. The former retrieves a collection of records from the database that match the query's conditions, while the latter only produces an aggregate, such as count, sum, average or max (min), of a particular attribute of these records. With reporting queries, the database is simply used as a data storage-retrieval tool. Many modern business intelligence applications, however, require ad hoc analytical queries with a rapid execution time. Users issuing these analytical queries are interested not in the actual records, but some statistics of them. This has therefore led to extensive research on how to perform aggregation queries efficiently. By augmenting a database index (very often a B-tree) with properly precomputed aggregates, aggregation queries can be answered efficiently at query time without going through the actual data records.

However, reporting and aggregation queries provide only two extremes for analyzing the data, by returning either all the records matching the query condition or one (or a few) single-valued aggregates. These simple aggregates are not expressive enough, and data analysts often need more insight into the data distribution. Consider the following queries:

(Q1) In a company's database: What is the distribution of salaries of all employees aged between 30 and 40?

(Q2) In a search engine's query logs: What are the most frequently queried keywords between May 1 and July 1, 2010?

The analyst issuing the query is perhaps not interested in listing all the records in the query range one by one, while probably not happy with a simple aggregate such as average or max, either. What would be nice is some summary on the data, which is more complex than the simple aggregates, yet still much smaller than the raw query results. Some useful summaries include the frequent items, the ϕ-quantiles for, say, $\phi = 0.1, 0.2, \ldots, 0.9$, a sketch (e.g., the Count-Min sketch [8] or the AMS sketch [4]), or some compressed data representations like wavelets. All these summaries are of central importance in massive data analysis, and have been extensively studied for offline and streaming data. Yet, to use the existing algorithms, one still has to first issue a reporting query to retrieve all query results, and then construct the desired summary afterward. This is clearly time-consuming and wasteful.

In this paper, we propose to add a native support for *summary queries* in a database index, such that a summary can be returned in time proportional to the size of the summary itself, not the size of the raw query results. The problem we consider can be defined more precisely as follows. Let $\mathcal{D}$ be a database containing N records. Each record $p \in \mathcal{D}$ is associated with a *query attribute* $A_q(p)$ and a *summary attribute* $A_s(p)$, drawing values possibly from different domains. A *summary query* specifies a range constraint $[q_1, q_2]$ on A_q and the database returns a summary on the A_s attribute of all records whose A_q attribute is within the range. For example, in the query (Q1) above, A_q is "age" and A_s is "salary". Note that A_s and A_q could be the same attribute, but it is more useful when they are different, as the analyst is exploring the relationship between two attributes. Our goal is to build an index on $\mathcal{D}$ so that a summary query can be answered efficiently. As with any indexing problem, the primary measures are the query time and the space the index uses. The index should also work in external memory, where it is stored in blocks of size B, and the query cost is measured in terms of the number of blocks accessed (I/Os). Finally, we also would like the index to support updates, i.e., insertion and deletion of records.

1.1 Related work on indexing for aggregation queries

In one dimension, most aggregates can be supported easily using a binary tree (a B-tree in external memory). At each internal node of the binary tree, we simply store the aggregate of all the records below the node. This way an aggregation query can be answered in $O(\log N)$ time ($O(\log_B N)$ I/Os in external memory).

In higher dimensions, the problem becomes more difficult and has been extensively studied in both the computational geometry and the database communities. Solutions are typically based on space-partitioning hierarchies, like partition trees, quadtrees and R-trees, where an internal node stores the aggregate for its subtree. There is a large body of work on spatial data structures; please refer to the survey by Agarwal and Erickson [2] and the book by Samet [26]. When the data space forms an array, the data cube [13] is an efficient structure for answering aggregation queries.

However, all the past research, whether in computational geometry or databases, has only considered queries that return simple aggregates like count, sum, max (min), and very recently top-k [1] and median [7, 18]. The problem of returning complex summaries has not been addressed.

1.2 Related work on (non-indexed) summaries

There is also a vast literature on various summaries in both the database and algorithms communities, motivated by the fact that simple aggregates cannot well capture the data distribution. These summaries, depending on the context and community, are also called *synopses*, *sketches*, or *compressed representations*. However, all past research has focused on how to construct a summary, either offline or in a streaming fashion, on the *entire* data set. No one has considered the indexing problem where the focus is to intelligently compute and store auxiliary information in the index at pre-computation time, so that a summary on a *requested subset* of the records in the database can be built quickly at query time. Since we cannot afford to look at all the requested records to build the summary at query time, this poses new challenges that past research cannot address: All existing construction algorithms need to at least read the data records once. The problem of how to maintain a summary as the underlying data changes, namely under insertions and deletions of records, has also been extensively studied. But this should not be confused with our dynamic index problem. The former maintains a single summary for the entire dynamic data set, while the latter aims at maintaining a dynamic structure from which a summary for any queried subset can be extracted, which is more general than the former. Of course for the former there often exist small-space solutions, while for the indexing problem, we cannot hope for sublinear space, as a query range may be small enough so that the summary degenerates to the raw query results.

Below we review some of the most fundamental and most studied summaries in the literature. Let D be a bag of items, and let $f_D(x)$ be the frequency of x in D.

Heavy hitters. A heavy hitters summary allows one to extract all frequent items approximately, i.e., for a user-specified $0 < \phi < 1$, it returns all items x with $f_D(x) > \phi|D|$ and no items with $f_D(x) < (\phi - \varepsilon)|D|$, while an item x with $(\phi - \varepsilon)|D| \le f_D(x) \le \phi|D|$ may or may not be returned. A heavy hitters summary of size $O(1/\varepsilon)$ can be constructed in one pass over D, using the MG algorithm [23] or the Space-Saving algorithm [22].

Sketches. Various sketches have been developed as a useful tool for summarizing massive data. In this paper, we consider the two most widely used ones: the *Count-Min sketch* [8] and the *AMS sketch* [4]. They summarize important information about D and can be used for a variety of purposes. Most notably, they can be used to estimate the join size of two data sets, with self-join size being a special case. Given the Count-Min sketches (resp. AMS sketches) of two data sets D_1 and D_2, we can estimate $|D_1 \bowtie D_2|$ within an additive error of $\varepsilon F_1(D_1)F_1(D_2)$ (resp. $\varepsilon\sqrt{F_2(D_1)F_2(D_2)}$) with probability at least $1 - \delta$ [3, 8], where F_k is the k-th frequency moment of D: $F_k(D) = \sum_x f_D^k(x)$. Note that $\sqrt{F_2(D)} \le F_1(D)$, so the error of the AMS sketch is no larger. However, its size is $O((1/\varepsilon^2)\log(1/\delta))$, which is larger than the Count-Min sketch's size $O((1/\varepsilon)\log(1/\delta))$, so they are not strictly comparable. Which one is better will depend on the skewness of the data sets. In particular, since $F_1(D) = |D|$, the error of the Count-Min sketch does not depend on the skewness of the data, but $F_2(D)$ could range from $|D|$ for uniform data to $|D|^2$ for highly skewed data.

Quantiles. The quantiles (a.k.a. the *order statistics*), which generalize the median, are important statistics about the data distribution. Recall that the ϕ-*quantile*, for $0 < \phi < 1$, of a set D of items from a totally ordered universe is the one ranked at $\phi|D|$ in D (for convenience, for the quantile problem it is usually assumed that there are no duplicates in D). A *quantile summary* contains enough information so that for any $0 < \phi < 1$, an ε-approximate ϕ-quantile can be extracted, i.e., the summary returns a ϕ'-quantile where $\phi - \varepsilon \le \phi' \le \phi + \varepsilon$. A quantile summary has size $O(1/\varepsilon)$, and can be easily computed by sorting D, and then taking the items ranked at $\varepsilon|D|, 2\varepsilon|D|, 3\varepsilon|D|, \ldots, |D|$.

Wavelets. *Wavelet representations* (or just *wavelets* for short) take a different approach to approximating the data distribution by borrowing ideas from signal processing. Suppose the records in D are drawn from an ordered universe

$[u] = \{1, \ldots, u\}$ and let $\mathbf{f}_D = (f_D(1), \ldots, f_D(u))$ be the frequency vector of D. Briefly speaking, in wavelet transformation we take u inner products $s_i = \langle \mathbf{f}_D, \mathbf{w}_i \rangle$ where $\mathbf{w}_i, i = 1, \ldots, u$ are the *wavelet basis vectors* (please refer to [12, 20] for details on wavelet basis vectors). The s_i's are called the *wavelet coefficients* of $\mathbf{f}_D$. If we kept all u wavelet coefficients, we would be able to reconstruct $\mathbf{f}_D$ exactly, but this would not be a "summary". The observation is that, for most real-world distributions, $\mathbf{f}_D$ yields few wavelet coefficients with large absolute values. Thus for a parameter k, even if we keep the k coefficients with the largest absolute values, and assume all the other coefficients are zero, we can still reconstruct $\mathbf{f}_D$ reasonably well. In fact, it is well known that among all the choices, retaining the k largest (in absolute value) coefficients minimizes the ℓ_2 error between the original $\mathbf{f}_D$ and the reconstructed one. Matias et al. [20] were the first to apply wavelet transformation to approximating data distributions. After that, wavelets have been extensively studied [9, 11, 12, 15, 21, 29], and have been shown to be highly effective at summarizing many real-world data distributions.

All the aforementioned work studies how to construct or maintain the summary on the given D. In our case, D is the A_s attributes of all records whose A_q attributes are within the query range. Our goal is to design an index so that the desired summary on D can be constructed efficiently without actually going through the elements of D.

1.3 Other related work

A few other lines of work also head to the general direction of addressing the gap between reporting all query results and returning some simple aggregates. Lin et al. [19] and Tao et al. [27] propose returning only a subset of the query results, called "representatives". But the "representatives" do not summarize the data as we do. They also only consider skyline queries. The line of work on *online aggregation* [16, 17] aims at producing a random sample of the query results at early stages of long-running queries, in particular, joins. A random sample indeed gives a rough approximation of the data distribution, but it is much less accurate than the summaries we consider: For heavy hitters and quantiles, a random sample of size $\Theta(1/\varepsilon^2)$ is needed [28] to achieve the same accuracy as the $O(1/\varepsilon)$-sized summaries we mentioned earlier; for estimating join sizes, a random sample of size $\Omega(\sqrt{N})$ is required to achieve a constant approximation, which is much worse than using the sketches [3]. Furthermore, the key difference is that they focus on query processing techniques for joins rather than indexing issues. Correlated aggregates [10] aim at exploring the relationship between two attributes. They are computed on one attribute subject to a certain condition on the other. However, this condition has to be specified in advance and the goal is to compute the aggregate in the streaming setting, thus the problem is fundamentally different from ours.

1.4 Our results

To take a unified approach we classify all the summaries mentioned in Section 1.2 into F_1-based ones and F_2-based ones. The former includes heavy hitters, the Count-Min sketch, and quantiles, all of which provide an error guarantee of the form $\varepsilon F_1(D)$ (note that an ε-approximate quantile is a value with a rank that is off by $\varepsilon F_1(D)$ from the correct rank). The latter includes the AMS sketch and wavelets, both of which provide an error guarantee related to $F_2(D)$.

In Section 2 we first design a baseline solution that works for all *decomposable* summaries. A summary is *decomposable* if given the summaries for t data sets (bags of elements) $D_1, \ldots, D_t$ with error parameter ε, we can combine them together into a summary on $D_1 \uplus \cdots \uplus D_t$ with error parameter $O(\varepsilon)$, where $\uplus$ denotes multiset addition. All the F_1 and F_2 based summaries have this property and thus can be plugged into this solution. Assuming that we can combine the summaries with cost linear to their total size, the resulting index has linear size and answers a summary query in $O(s_\varepsilon \log N)$ time, where s_ε is the size of the summary returned. It also works in external memory, with the query cost being $O(\frac{s_\varepsilon}{B} \log N)$ I/Os if $s_\varepsilon \geq B$ and $O(\log N / \log(B/s_\varepsilon))$ I/Os if $s_\varepsilon < B$. Note that this decomposable property has been exploited in many other works on maintaining summaries in the streaming context [5, 6, 8].

In Section 3 we improve upon this baseline solution by identifying another, stronger decomposable property of the F_1 based summaries, which we call *exponentially decomposable*. The size of the index remains linear, while its query cost improves to $O(\log N + s_\varepsilon)$. In external memory, the query cost is $O(\log_B N + s_\varepsilon/B)$ I/Os. This resembles the classical B-tree query cost, which includes an $O(\log_B N)$ search cost and an "output" cost of $O(s_\varepsilon/B)$, whereas the output in our case is a summary of size s_ε. This is clearly optimal (in the comparison model). For not-too-large summaries $s_\varepsilon = O(B)$, the query cost becomes just $O(\log_B N)$, the same as that for a simple aggregation query or a lookup on a B-tree.

In Section 4, we demonstrate how various summaries have the desired decomposable or exponentially decomposable property and thus can be plugged into our indexes. Finally we show how to support updates in Section 5.

2. A BASELINE SOLUTION

In this and the next section, we will describe our structures without instantiating with any particular summary. Instead we just use "ε-summary" as a placeholder for any summary with error parameter ε. Let $\mathcal{S}(\varepsilon, D)$ denote an ε-summary on data set D. We use s_ε to denote the size of an ε-summary[1].

Internal memory structure. Based on the decomposable property of a summary, a baseline solution can be designed using standard techniques. We first describe the internal memory structure. Sort all the N data records in the database on the A_q attribute and partition them into N/s_ε groups, each of size s_ε. Then we build a binary tree $\mathcal{T}$ on top of these groups, where each leaf (called a *fat leaf*) stores a group of s_ε records. For each internal node u of $\mathcal{T}$, let $\mathcal{T}_u$ denote the subtree of $\mathcal{T}$ rooted at u. We attach to u an ε-summary on the A_s attribute of all records stored in the subtree below u. Since each ε-summary has size s_ε and the number of internal nodes is $O(N/s_\varepsilon)$, the total size of the structure is $O(N)$. To answer a query $[q_1, q_2]$, we do a search on $\mathcal{T}$. It is well known that any range $[q_1, q_2]$ can be decomposed into $O(\log(N/s_\varepsilon))$ disjoint *canonical* subtrees $\mathcal{T}_u$, plus at most two fat leaves that may partially overlap. We retrieve the ε-summaries attached to the roots of these

[1]Strictly speaking we should write $s_{\varepsilon,D}$. But as most ε-summaries have sizes independent of D, we drop the subscript D for brevity.

subtrees. For each of the fat leaves, we simply read all the s_ε records stored there. Then we combine all of them into an $O(\varepsilon)$-summary for the entire query using the decomposable property. We can adjust ε by a constant factor in the construction to ensure that the output is an ε-summary. The total query time is thus the time required to combine the $O(\log(N/s_\varepsilon))$ summaries. For the Count-Min sketch and AMS sketch, the combining time is linear in the total size of the summaries, so the query time is $O(s_\varepsilon \log N)$. For the quantile summary and heavy hitters summary the query time becomes $O(s_\varepsilon \log N \log\log N)$[2] as we need to merge $O(\log(N/s_\varepsilon))$ sorted lists (Details in Section 4).

External memory index. The baseline solution easily extends to external memory. If $s_\varepsilon \geq B$, then each internal node and fat leaf occupies $\Theta(s_\varepsilon/B)$ blocks, so we can simply store each of them separately. The space is still linear and we load $O(\log N)$ nodes on each query. The query cost becomes $O(\frac{s_\varepsilon}{B}\log N)$ I/Os for the Count-Min and AMS sketch and $O(\frac{s_\varepsilon}{B}\log N \log_{M/B}\log N)$ I/Os for the quantile and heavy hitters summary.

For $s_\varepsilon < B$, each node occupies a fraction of a block, and we can pack multiple nodes in one block. We use a standard B-tree blocking of the tree $\mathcal{T}$ where each block contains $\Theta(B/s_\varepsilon)$ nodes, except possibly the root block. Thus each block stores a subtree of height $\Theta(\log(B/s_\varepsilon))$ of $\mathcal{T}$. Then standard analysis shows that the nodes we need to access are stored in $O(\log N/\log(B/s_\varepsilon))$ blocks. This implies a query cost of $O(\log_{B/s_\varepsilon} N)$ I/Os for the Count-Min and AMS sketch and $O(\log_{B/s_\varepsilon} N \log_{M/B}(\log_{B/s_\varepsilon} N))$ I/Os for the quantile and heavy hitters summary.

3. OPTIMAL INDEXING FOR F_1 BASED SUMMARIES

The baseline solution of the previous section is not that impressive: Its "output" term has an extra $O(\log N)$ factor; in external memory, we are missing the ideal $O(\log_B N)$ term which is the main benefit of block accesses.

The main bottleneck in the baseline solution is not the search cost, but the fact that we need to assemble $O(\log N)$ summaries, each of size s_ε. In the absence of additional properties of the summary, it is impossible to make further improvement. Fortunately, we observe that many of the F_1 based summaries have what we call the *exponentially decomposable* property, which allows us to assemble summaries of exponentially decreasing sizes. This turns out to be the key to optimality for indexing these summaries.

Definition 1 (Exponentially decomposable). *For $0 < \alpha < 1$, a summary $\mathcal{S}$ is α-exponentially decomposable if there exists a constant $c > 1$, such that for any t multisets $D_1, \dots, D_t$ with their sizes satisfying $F_1(D_i) \leq \alpha^{i-1}F_1(D_1)$ for $i = 1, \dots, t$, given $\mathcal{S}(\varepsilon, D_1), \mathcal{S}(c\varepsilon, D_2) \dots, \mathcal{S}(c^{t-1}\varepsilon, D_t)$, (1) we can construct an $O(\varepsilon)$-summary for $D_1 \uplus \cdots \uplus D_t$; (2) the total size of $\mathcal{S}(\varepsilon, D_1), \dots, \mathcal{S}(c^{t-1}\varepsilon, D_t)$ is $O(s_\varepsilon)$ and they can be combined in $O(s_\varepsilon)$ time; and (3) for any multiset D, the total size of $\mathcal{S}(\varepsilon, D), \dots, \mathcal{S}(c^{t-1}\varepsilon, D)$ is $O(s_\varepsilon)$.*

Intuitively, since an F_1 based summary $\mathcal{S}(\varepsilon, D)$ provides an error bound of $\varepsilon|D|$, the total error from $\mathcal{S}(\varepsilon, D_1)$, $\mathcal{S}(c\varepsilon, D_2)$, $\dots$, $\mathcal{S}(c^{t-1}\varepsilon, D_t)$ is

$$\begin{aligned} & \varepsilon|D_1| + c\varepsilon|D_2| + \cdots + c^{t-1}\varepsilon|D_t| \\ \leq\ & \varepsilon|D_1| + (c\alpha)\varepsilon|D_1| + \cdots + (c\alpha)^{t-1}\varepsilon|D_1|. \end{aligned}$$

If we choose c such that $c\alpha < 1$, then the error is bounded by $O(\varepsilon|D_1|)$, satisfying (1). Meanwhile, the F_1 based summaries usually have size $s_\varepsilon = \Theta(1/\varepsilon)$, so (2) and (3) can be satisfied, too. In Section 4 we will formally prove the α-exponentially decomposable property for all the F_1 based summaries mentioned in Section 1.2.

3.1 Optimal internal memory structure

Let $\mathcal{T}$ be the binary tree built on the A_q attribute as in the previous section. Without loss of generality we assume $\mathcal{T}$ is a complete balanced binary tree; otherwise we can always add at most N dummy records to make N/s_ε a power of 2 so that $\mathcal{T}$ is complete.

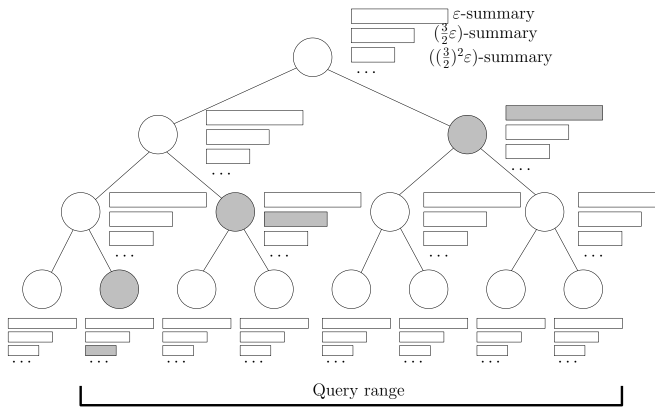
We first define some notation on $\mathcal{T}$. We use $\mathcal{S}(\varepsilon, u)$ to denote the ε-summary on the A_s attribute of all records stored in u's subtree. Fix an internal node u and a descendant v of u, let $\mathcal{P}(u, v)$ to be the set of nodes on the path from u to v, excluding u. Define the *left sibling set* of $\mathcal{P}(u, v)$ to be $\mathcal{L}(u, v) = \{w \mid w \text{ is a left child and has a right sibling} \in \mathcal{P}(u, v)\}$ and similarly the *right sibling set* of $\mathcal{P}(u, v)$ to be $\mathcal{R}(u, v) = \{w \mid w \text{ is a right child and has a left sibling} \in \mathcal{P}(u, v)\}$. To answer a query $[q_1, q_2]$, we first locate the two fat leaves a and b in $\mathcal{T}$ that contain q_1 and q_2, respectively. Let u be the lowest common ancestor of a and b. We call $\mathcal{P}(u, a)$ and $\mathcal{P}(u, b)$ the left and respectively the right query path. We observe that the subtrees rooted at the nodes in $\mathcal{R}(u, a) \cup \mathcal{L}(u, b)$ make up the canonical set for the query range $[q_1, q_2]$.

Focusing on $\mathcal{R}(u, a)$, let $w_1, \dots, w_t$ be the nodes of $\mathcal{R}(u, a)$ and let $d_1 < \dots < d_t$ denote their depths in $\mathcal{T}$ (the root of $\mathcal{T}$ is said to be at depth 0). Since $\mathcal{T}$ is a balanced binary tree, we have $F_1(w_i) \leq (1/2)^{d_i - d_1}F_1(w_1)$ for $i = 1, \dots, t$. Here we use $F_1(w)$ to denote the first frequency moment (i.e., size) of the point set rooted at node w. Thus, if the summary is $(1/2)$-exponentially decomposable, and we have $\mathcal{S}(c^{d_i - d_1}\varepsilon, w_i)$ for $i = 1, \dots, t$ at our disposal, we can combine them and form an $O(\varepsilon)$-summary for all the data covered by $w_1, \dots, w_t$. We do the same for $\mathcal{L}(u, b)$. Finally, the two fat leaves can always supply the exact data (it is a summary with no error) of size $O(s_\varepsilon)$ in the query range. Plus the initial $O(\log N)$ search cost for locating $\mathcal{R}(u, a)$ and $\mathcal{L}(u, b)$, the query time now improves to the optimal $O(\log N + s_\varepsilon)$.

It only remains to show how to supply $\mathcal{S}(c^{d_i - d_1}\varepsilon, w_i)$ for each of the w_i's. In fact, we can afford to attach to each node $u \in \mathcal{T}$ all the summaries: $\mathcal{S}(\varepsilon, u), \mathcal{S}(c\varepsilon, u), \dots \mathcal{S}(c^q\varepsilon, u)$ where q is an integer such that $s_{c^q\varepsilon} = O(1)$. Nicely, these summaries still have total size $O(s_\varepsilon)$ by the exponentially decomposable property, thus the space required by each node is still $O(s_\varepsilon)$ as in the previous section, and the total space remains linear. A schematic illustration of the overall structure is shown in Figure 1.

Theorem 1. *For any $(1/2)$-exponentially decomposable summary, a database $\mathcal{D}$ of N records can be stored in an internal memory structure of linear size so that a summary query can be answered in $O(\log N + s_\varepsilon)$ time.*

[2]In fact, an alternative solution achieves query time $O(s_\varepsilon \log N/\log\log N)$ by issuing s_ε range-quantile queries to the data structure in [7], but this solution does not work in external memory.



ε-summary
$(\frac{3}{2}\varepsilon)$-summary
$((\frac{3}{2})^2\varepsilon)$-summary
Query range

Figure 1: A schematic illustration of our internal memory structure. The grayed nodes form the canonical decomposition of the query range, and the grayed summaries are those we combine into the final summary for the queried data. In this example we use $c = \frac{3}{2}$.

3.2 Optimal external memory indexing

In this section we show how to achieve the $O(\log_B N + s_\varepsilon/B)$-I/O query cost in external memory still with linear space. Here, the difficulty that we need to assemble $O(\log N)$ summaries lingers. In internal memory, we managed to get around it by the exponentially decomposable property so that the total size of these summaries is $O(s_\varepsilon)$. However, they still reside in $O(\log N)$ separate nodes. If we still use a standard B-tree blocking, for $s_\varepsilon \geq B$ we need to access $\Omega(\log N)$ blocks; for $s_\varepsilon < B$, we need to access $\Omega(\log N/\log(B/s_\varepsilon))$ blocks, neither of which is optimal. Below we first show how to achieve the optimal query cost by increasing the space to super-linear, then propose a packed structure to reduce the space back to linear.

Consider an internal node u and one of its descendants v. Let the sibling sets $\mathcal{R}(u,v)$ and $\mathcal{L}(u,v)$ be as previously defined. In the following we only describe how to handle the $\mathcal{R}(u,v)$'s; we will do the same for the $\mathcal{L}(u,v)$'s. Suppose $\mathcal{R}(u,v)$ contains nodes $w_1, \dots, w_t$ at depths $d_1, \dots, d_t$. We define the *summary set* for $\mathcal{R}(u,v)$ with error parameter ε to be

$$\mathcal{RS}(u,v,\varepsilon) = \{\mathcal{S}(\varepsilon, w_1), \mathcal{S}(c^{d_2-d_1}\varepsilon, w_2), \dots, \mathcal{S}(c^{d_t-d_1}\varepsilon, w_t)\}.$$

The following two facts easily follow from the exponentially decomposable property.

FACT 1. *The total size of the summaries in $\mathcal{RS}(u,v,\varepsilon)$ is $O(s_\varepsilon)$;*

FACT 2. *The total size of all the summary sets $\mathcal{RS}(u,v,\varepsilon)$, $\mathcal{RS}(u,v,c\varepsilon)$,..., $\mathcal{RS}(u,v,c^t\varepsilon)$ is $O(s_\varepsilon)$.*

The indexing structure. We first build the binary tree $\mathcal{T}$ as before with a fat leaf size of s_ε. Before attaching any summaries, we block $\mathcal{T}$ in a standard B-tree fashion so that each block stores a subtree of $\mathcal{T}$ of size $\Theta(B)$, except possibly the root block which may contain 2 to B nodes of $\mathcal{T}$. The resulting blocked tree is essentially a B-tree where each leaf occupies $O(s_\varepsilon/B)$ blocks and each internal node occupies 1 block. Please see Figure 2 for an example of the standard B-tree blocking.

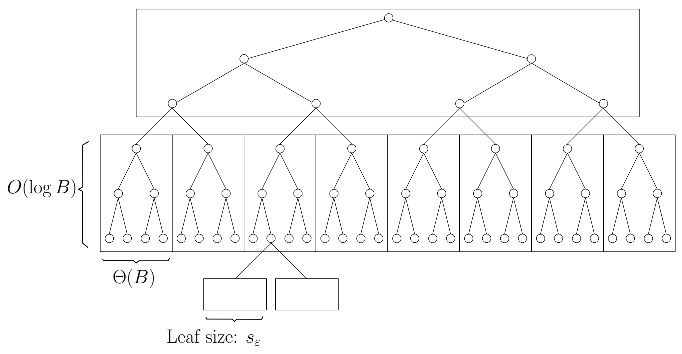
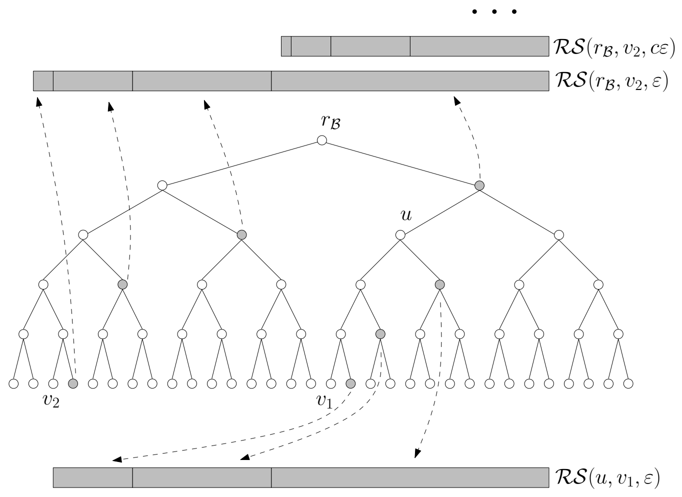


Figure 2: The standard B-tree blocking of a binary tree.



$\mathcal{RS}(r_\mathcal{B}, v_2, c\varepsilon)$
$\mathcal{RS}(r_\mathcal{B}, v_2, \varepsilon)$
$r_\mathcal{B}$
u
v_2
v_1
$\mathcal{RS}(u, v_1, \varepsilon)$

Figure 3: The summaries we store for an internal block $\mathcal{B}$.

Consider an internal block $\mathcal{B}$ in the B-tree. Below we describe the additional structures we attach to $\mathcal{B}$. Let $\mathcal{T}_\mathcal{B}$ be the binary subtree of $\mathcal{T}$ stored in $\mathcal{B}$ and let $r_\mathcal{B}$ be the root of $\mathcal{T}_\mathcal{B}$. To achieve the optimal query cost, the summaries attached to the nodes of $\mathcal{T}_\mathcal{B}$ that we need to retrieve for answering any query must be stored consecutively, or in at most $O(1)$ consecutive chunks. Therefore, the idea is to store all the summaries for a query path in $\mathcal{T}_\mathcal{B}$ together, which is the reason we introduced the summary set $\mathcal{RS}(u,v,\varepsilon)$. The detailed structures that we attach to $\mathcal{B}$ are as follows:

1. For each internal node $u \in \mathcal{T}_\mathcal{B}$ and each leaf v in u's subtree in $\mathcal{T}_\mathcal{B}$, we store all summaries in $\mathcal{RS}(u,v,\varepsilon)$ sequentially.

2. For each leaf v, we store the summaries in $\mathcal{RS}(r_\mathcal{B}, v, c^j\varepsilon)$ sequentially, for all $j = 0, \dots, q$. Recall that q is an integer such that $s_{c^q\varepsilon} = O(1)$.

3. For the root $r_\mathcal{B}$, we store $\mathcal{S}(c^j\varepsilon, r_\mathcal{B})$ for $j = 0, \dots, q$.

An illustration of the first and the second type of structures is shown in Figure 3. The size of the structure can be determined as follow:

1. For each leaf $v \in \mathcal{T}_\mathcal{B}$, there are at most $O(\log B)$ ancestors of v, so there are in total $O(B \log B)$ such pairs (u, v). For each pair we use $O(s_\varepsilon)$ space, so the space usage is $O(s_\varepsilon B \log B)$.

2. For each leaf $v \in \mathcal{T}_\mathcal{B}$ we use $O(s_\varepsilon)$ space, so the space usage is $O(s_\varepsilon B)$.

3. For the root $r_\mathcal{B}$, the space usage is $O(s_\varepsilon)$.

Summing up the above cases, the space for storing the summaries of any internal block $\mathcal{B}$ is $O(s_\varepsilon B \log B)$. Note that each internal block has fanout $\Theta(B)$, and each leaf has size $\Theta(s_\varepsilon)$, so there are in total at most $O(N/(Bs_\varepsilon))$ internal blocks, and thus the total space usage is $O(N \log B)$. Next we show that this structure can indeed be used to answer queries in the optimal $O(\log_B N + s_\varepsilon/B)$ I/Os.

Query procedure. Given a query range $[q_1, q_2]$, let a and b be the two leaves containing q_1 and q_2, respectively. We focus on how to retrieve the necessary summaries for the right sibling set $\mathcal{R}(u, a)$, where u is the lowest common ancestor of a and b; the left sibling set $\mathcal{L}(u, b)$ can be handled symmetrically. By the previous analysis, we need exactly the summaries in $\mathcal{RS}(u, a, \varepsilon)$. Recall that $\mathcal{R}(u, a)$ are the right siblings of the left query path $\mathcal{P}(u, a)$. Let $\mathcal{B}_0, \ldots, \mathcal{B}_l$ be the blocks that $\mathcal{P}(u, a)$ intersects from u to a. The path $\mathcal{P}(u, a)$ is partitioned into $l + 1$ segments by these $l + 1$ blocks. Let $\mathcal{P}(u, v_0), \mathcal{P}(r_1, v_1), \ldots, \mathcal{P}(r_l, v_l = a)$ be the $l + 1$ segments, with r_i being the root of the binary tree $\mathcal{T}_{\mathcal{B}_i}$ in block $\mathcal{B}_i$ and v_i being a leaf of $\mathcal{T}_{\mathcal{B}_i}$, $i = 0, \ldots, l$. Let $w_1, \ldots, w_t$ be the nodes in $\mathcal{R}(u, a)$, at depths $d_1, \ldots, d_t$ of $\mathcal{T}$. We claim that w_i is either a node of $\mathcal{T}_{\mathcal{B}_k}$ for some $k \in \{0, \ldots, l\}$, or a right sibling of r_k for some $k \in \{0, \ldots, l\}$, which makes w_i a root of some other block. This is because by the definition of $\mathcal{R}(u, a)$, we know that w_i is a right child whose left sibling is in some $\mathcal{B}_k$. If w_i is not in $\mathcal{B}_k$, it must be the root of some other block. Recall that we need to retrieve $\mathcal{S}(c^{d_i - d_1}\varepsilon, w_i)$ for $i = 1, \ldots, t$. Below we show how this can be done efficiently using our structure.

For the w_i's in the first block $\mathcal{B}_0$, since we have stored all summaries in $\mathcal{RS}(u, v_0, \varepsilon)$ sequentially for $\mathcal{B}_0$ (case 1.), they can be retrieved in $O(1 + s_\varepsilon/B)$ I/Os.

For any w_i being the root of some other block $\mathcal{B}'$ not on the path $\mathcal{B}_0, \ldots, \mathcal{B}_l$, since we have stored the summaries $\mathcal{S}(c^j\varepsilon, w_i)$ for $j = 0, \ldots, q$ for every block (case 3.), the required summary $\mathcal{S}(c^{d_i - d_1}\varepsilon, w_i)$ can be retrieved in $O(1 + s_{c^{d_i - d_1}\varepsilon}/B)$ I/Os. Note that the number of such w_i's is bounded by $O(\log_B N)$, so the total cost for retrieving summaries for these nodes is at most $O(\log_B N + s_\varepsilon/B)$ I/Os.

The rest of the w_i's are in $\mathcal{B}_1, \ldots, \mathcal{B}_l$. Consider each $\mathcal{B}_k, k = 1, \ldots, l$. Recall that the segment of the path $\mathcal{P}(u, a)$ in $\mathcal{B}_k$ is $\mathcal{P}(r_k, v_k)$, and the w_i's in $\mathcal{B}_k$ are exactly $\mathcal{R}(r_k, v_k)$. We have stored $\mathcal{RS}(r_k, v_k, c^j\varepsilon)$ for $\mathcal{B}_k$ for all j (case 2.), so no matter at which relative depths $d_i - d_1$ the nodes in $\mathcal{R}(r_k, v_k)$ start and end, we can always find the required summary set. Retrieving the desired summary set takes $O\left(1 + s_{c^{d' - d_1}\varepsilon}/B\right)$ I/Os, where d' is the depth of the highest node in $\mathcal{R}(r_k, v_k)$. Summing over all blocks $\mathcal{B}_1, \ldots, \mathcal{B}_l$, the total cost is $O(\log_B N + s_\varepsilon/B)$ I/Os.

Reducing the size to linear. The structure above has a super-linear size $O(N \log B)$. Next we show how to reduce its size back to $O(N)$ while not affecting the optimal query cost.

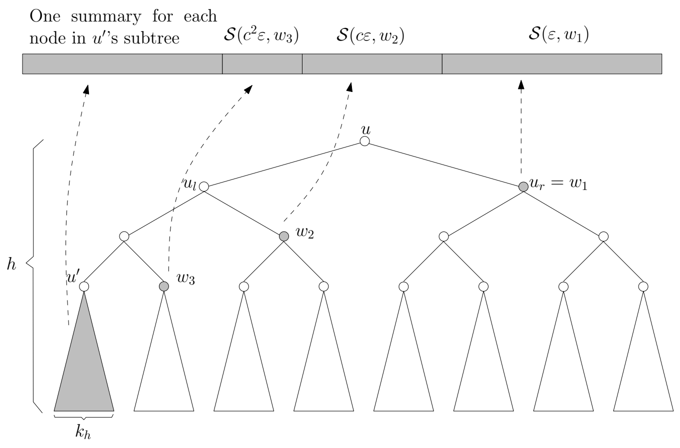

One summary for each node in u''s subtree
$\mathcal{S}(c^2\varepsilon, w_3)$
$\mathcal{S}(c\varepsilon, w_2)$
$\mathcal{S}(\varepsilon, w_1)$
u
u_l
$u_r = w_1$
w_2
h
u'
w_3
k_h

Figure 4: A schematic illustration of our packed structure.

Observe that the $\log B$ factor comes from case 1., where we store $\mathcal{RS}(u, v, \varepsilon)$ for each internal node u and each leaf v in u's subtree in u's block $\mathcal{B}$. Focus on $\mathcal{B}$ and the binary tree $\mathcal{T}_\mathcal{B}$ stored in it. Abusing notation, we use $\mathcal{T}_u$ to denote the subtree rooted at u in $\mathcal{T}_\mathcal{B}$. Assume $\mathcal{T}_u$ has height h (in $\mathcal{T}_\mathcal{B}$). Our idea is to pack the $\mathcal{RS}(u, v, \varepsilon)$'s for some leaves $v \in \mathcal{T}_u$ to reduce the space usage. Let u_l and u_r be the left and right child of u, respectively. The first observation is that we only need to store $\mathcal{RS}(u, v, \varepsilon)$ for each leaf v in u_l's subtree. This is because for any leaf v in u_r's subtree, the sibling set $\mathcal{R}(u, v)$ is the same as $\mathcal{R}(u_r, v)$, so $\mathcal{RS}(u, v, \varepsilon) = \mathcal{RS}(u_r, v, \varepsilon)$, which will be stored when considering u_r in place of u. For any leaf v in u_l's subtree, observe that the highest node in $\mathcal{R}(u, v)$ is u_r. This means for a node $w \in \mathcal{R}(u, v)$ with height i in tree $\mathcal{T}_u$, the summary for w in $\mathcal{RS}(u, v, \varepsilon)$ is $\mathcal{S}(c^{h-i-1}\varepsilon, w)$. Let u' be an internal node in u_l's subtree, and suppose u' has k_h leaves below it. We will decide later the value of k_h and, thus, the height $\log k_h$ at which u' is chosen (the leaves are defined to be at height 0). We do the following for each u' at height $\log k_h$ in u_l's subtree. Instead of storing the summary set $\mathcal{RS}(u, v, \varepsilon)$ for each leaf v in u''s subtree, we store $\mathcal{RS}(u, u', \varepsilon)$, which is the common prefix of all the $\mathcal{RS}(u, v, \varepsilon)$'s, together with a summary for each of the nodes in u''s subtree. More precisely, for each node w in u''s subtree, if its height is i, we store a summary $\mathcal{S}(c^{h-i-1}\varepsilon, w)$. All these summaries below u' are stored sequentially. A schematic illustration of our packed structure is shown in Figure 4.

Recall that all the summary sets we store in case 1. are used to cover the top portion of the query path $\mathcal{P}(u, v_0)$ in block $\mathcal{B}_0$, i.e., $\mathcal{RS}(u, v_0, \varepsilon)$. Clearly the packed structure still serves this purpose: We first find the u' which has v_0 as one of its descendants. Then we load $\mathcal{RS}(u, u', \varepsilon)$, followed by the summaries $\mathcal{S}(c^{h-i-1}, w)$ required in $\mathcal{RS}(u, v_0, \varepsilon)$. Loading $\mathcal{RS}(u, u', \varepsilon)$ still takes $O(1 + s_\varepsilon/B)$ I/Os, but loading the remaining individual summaries may incur many I/Os since they may not be stored sequentially. Nevertheless, if we ensure that all the individual summaries below u' have total size $O(s_\varepsilon)$, then loading any subset of them does not take more than $O(1 + s_\varepsilon/B)$ I/Os. Note that there are $k_h/2^i$ nodes at height i in u''s subtree, the total size of all summaries below u' is

$$\sum_{i=0}^{\log k_h} \frac{k_h}{2^i} s_{c^{h-i-1}\varepsilon}. \qquad (1)$$

Thus it is sufficient to choose k_h such that (1) is $\Theta(s_\varepsilon)$. Note that such a k_h always exists[3]: When $k_h = 1$, (1) is $s_{c^{h-1}\varepsilon} = O(s_\varepsilon)$; when k_h takes the maximum possible value $k_h = 2^{h-1}$, the last term (when $i = h$) in the summation of (1) is s_ε, so (1) is at least $\Omega(s_\varepsilon)$; every time k_h doubles, (1) increases by at most $O(s_\varepsilon)$.

It only remains to show that by employing the packed structure, the space usage for a block is indeed $O(Bs_\varepsilon)$. For a node u at height h in $\mathcal{T}_\mathcal{B}$, the number of u''s at height $\log k_h$ under u is $2^h/k_h$. For each such u', storing $\mathcal{RS}(u, u', \varepsilon)$, as well as all the individual summaries below u', takes $O(s_\varepsilon)$ space. So the space required for node u is $O(2^h s_\varepsilon/k_h)$. There are $O(B/2^h)$ nodes u at height h. Thus the total space required is

$$O\left(\sum_{h=1}^{\log B} 2^h s_\varepsilon/k_h \cdot B/2^h\right) = O\left(\sum_{h=1}^{\log B} Bs_\varepsilon/k_h\right).$$

Note that the choice of k_h implies that

$$s_\varepsilon/k_h = O\left(\sum_{i=0}^{\log k_h} \frac{1}{2^i} s_{c^{h-i-1}\varepsilon}\right) = O\left(\sum_{i=0}^{h-1} \frac{1}{2^i} s_{c^{h-i-1}\varepsilon}\right),$$

so the total size of the packed structures in $\mathcal{B}$ is bounded by

$$\begin{aligned}
\sum_{h=1}^{\log B} Bs_\varepsilon/k_h &\le B \sum_{h=0}^{\log B}\sum_{i=0}^{h-1} \frac{1}{2^i} s_{c^{h-i-1}\varepsilon} \\
&= B \sum_{h=0}^{\log B}\sum_{i=0}^{h-1} \frac{1}{2^{h-i-1}} s_{c^i\varepsilon} \\
&\le B \sum_{i=0}^{\log B} s_{c^i\varepsilon} \sum_{h=i}^{\log B} \frac{1}{2^{h-i-1}} \\
&\le 2B \sum_{i=0}^{\log B} s_{c^i\varepsilon} \\
&= O(Bs_\varepsilon).
\end{aligned}$$

THEOREM 2. *For any (1/2)-exponentially decomposable summary, a database $\mathcal{D}$ of N records can be stored in an external memory index of linear size so that a summary query can be answered in $O(\log_B N + s_\varepsilon/B)$ I/Os.*

Remark. One technical subtlety is that the $O(s_\varepsilon)$ combining time in internal memory does not guarantee that we can combine the $O(\log N)$ summaries in $O(s_\varepsilon/B)$ I/Os in external memory. However if the merging algorithm only makes linear scans on the summaries, then this is not a problem, as we shall see in Section 4.

4. SUMMARIES

In this section we demonstrate the decomposable or exponentially decomposable properties for the summaries mentioned in Section 1.2. Thus, they can be used in our indexes in Section 2 and 3.

[3]We define k_h in this implicit way for its generality. When instantiating into specific summaries, there are often closed forms for k_h. For example when $s_\varepsilon = \Theta(1/\varepsilon)$ and $1 < c < 2$, $k_h = \Theta(c^h)$.

4.1 Heavy hitters

Given a multiset D, let $f_D(x)$ be the frequency of x in D. The MG summary [23] with error parameter ε consists of $s_\varepsilon = 1/\varepsilon$ items and their associated counters. For any item x in the counter set, the MG summary maintains an estimated count $\hat{f}_D(x)$ such that $f_D(x) - \varepsilon F_1(D) \le \hat{f}_D(x) \le f_D(x)$; for any item x not in the counter set, it is guaranteed that $f_D(x) \le \varepsilon F_1(D)$. Thus in either case, the MG summary provides an additive $\varepsilon F_1(D)$ error: $f_D(x) - \varepsilon F_1(D) \le \hat{f}_D(x) \le f_D(x)$ for any x. The SpaceSaving summary is very similar to the MG summary except that the SpaceSaving summary provides an $\hat{f}_D(x)$ overestimating $f_D(x)$: $f_D(x) \le \hat{f}_D(x) < f_D(x) + \varepsilon F_1(D)$. Thus they clearly solve the heavy hitters problem.

The MG summary is clearly decomposable. Below we show that it is also α-exponentially decomposable for any $0 < \alpha < 1$. The same proof also works for the SpaceSaving summary.

Consider t multisets $D_1, \dots, D_t$ with $F_1(D_i) \le \alpha^{i-1} F_1(D_1)$ for $i = 1, \dots, t$. We set $c = 1/\sqrt{\alpha} > 1$. Given a series of MG summaries $\mathcal{S}(\varepsilon, D_1)$, $\mathcal{S}(c\varepsilon, D_2)$, $\dots$, $\mathcal{S}(c^{t-1}\varepsilon, D_t)$, we combine them by adding up the counters for the same item. Note that the total size of these summaries is bounded by

$$\sum_{j=0}^{t-1} s_{c^j\varepsilon} = \sum_{j=0}^{t-1} \frac{1}{c^j \varepsilon} = O(1/\varepsilon) = O(s_\varepsilon).$$

In order to analyze the error in the combined summary, let $f_j(x)$ denote the true frequency of item x in D_j and $\hat{f}_j(x)$ be the estimator of $f_j(x)$ in $\mathcal{S}(c^{j-1}\varepsilon, D_j)$. The combined summary uses $\sum_{j=1}^t \hat{f}_j(x)$ to estimate the true frequency of x, which is $\sum_{j=1}^t f_j(x)$. Note that

$$f_j(x) \ge \hat{f}_j(x) \ge f_j(x) - c^{j-1}\varepsilon F_1(D_j)$$

for $j = 1, \dots, t$. Summing up the first inequality over all j yields $\sum_{j=1}^t f_j(x) \ge \sum_{j=1}^t \hat{f}_j(x)$. For the second inequality, we have

$$\begin{aligned}
\sum_{j=1}^t \hat{f}_j(x) &\ge \sum_{j=1}^t f_j(x) - \sum_{j=1}^t c^{j-1}\varepsilon F_1(D_j) \\
&\ge \sum_{j=1}^t f_j(x) - \sum_{j=1}^t (\frac{\alpha}{\sqrt{\alpha}})^{j-1} \varepsilon F_1(D_1) \\
&\ge \sum_{j=1}^t f_j(x) - \varepsilon F_1(D_1) \sum_{j=1}^t (\sqrt{\alpha})^{j-1} \\
&= \sum_{j=1}^t f_j(x) - O(\varepsilon F_1(D_1)).
\end{aligned}$$

Therefore the error bound is $O(\varepsilon F_1(D_1)) = O(\varepsilon(F_1(D_1 \uplus \cdots \uplus D_t))$.

To combine the summaries we require that each summary maintains its (item, counter) pairs in the increasing order of items (we impose an arbitrary ordering if the items are from an unordered domain). In this case each summary can be viewed as a sorted list and we can merge the t sorted lists into a single list, where the counters for the same item are added up. Note that if each summary is of size s_ε, then we need to employ a t-way merging algorithm and it takes $O(s_\varepsilon t \log t)$ time in internal memory and $O(\frac{s_\varepsilon t}{B} \log_{M/B} t)$ I/Os in external memory. However, when the sizes of the t summaries form a geometrically decreasing sequence, we can repeatedly perform two-way merges in a bottom-up fashion with linear total cost. The merging algorithm starts with an empty list, at step i, it merges the current list with the summary $\mathcal{S}(\varepsilon_{t+1-i}, D_{t+1-i})$. Note that in this process every counter of $\mathcal{S}(\varepsilon_j, D_j)$ is merged j times, but since the size of $\mathcal{S}(\varepsilon_j, D_j)$ is $\frac{1}{c^{j-1}\varepsilon}$, the total running time is bounded by

$$\sum_{j=1}^{t} \frac{j}{c^{j-1}\varepsilon} = O\left(\frac{1}{\varepsilon}\right) = O(s_\varepsilon).$$

In external memory we can perform the same trick and achieve the $O(s_\varepsilon/B)$ I/O bound if the smallest summary $\mathcal{S}(c^{t-1}\varepsilon, D_t)$ has size $\frac{1}{c^{t-1}\varepsilon} > B$; otherwise we can take the smallest k summaries, where k is the maximum number such that the smallest k summaries can fit in one block, and merge them in the main memory. In either case, we can merge the t summaries in s_ε/B I/Os.

4.2 Quantiles

Recall that in the ε-approximate quantile problem, we are given a set D of N items from a totally ordered universe, and the goal is to have a summary $\mathcal{S}(\varepsilon, D)$ from which for any $0 < \phi < 1$, a record with rank in $[(\phi-\varepsilon)N, (\phi+\varepsilon)N]$ can be extracted. It is easy to obtain a quantile summary of size $O(1/\varepsilon)$: We simply sort D and take an item every εN consecutive items. Given any rank $r = \phi N$, there is always an element within rank $[r - \varepsilon N, r + \varepsilon N]$.

Below we show that quantile summaries are α-exponentially decomposable. Suppose we are given a series of such quantile summaries $\mathcal{S}(\varepsilon_1, D_1), \mathcal{S}(\varepsilon_2, D_2), \ldots, \mathcal{S}(\varepsilon_t, D_t)$, for data sets $D_1, \ldots, D_t$. We combine them by sorting all the items in these summaries. We claim this forms an approximate quantile summary for $D = D_1 \cup \cdots \cup D_t$ with error at most $\sum_{j=1}^{t} \varepsilon_j F_1(D_j)$, that is, given a rank r, we can find an item in the combined summary whose rank is in $[r - \sum_{j=1}^{t} \varepsilon_j F_1(D_j), r + \sum_{j=1}^{t} \varepsilon_j F_1(D_j)]$ in D. For an element x in the combined summary, let y_j and z_j be the two consecutive elements in $\mathcal{S}(\varepsilon_j, D_j)$ such that $y_j \le x \le z_j$. We define $r_j^{\min}(x)$ to be the rank of y_j in D_j and $r_j^{\max}(x)$ to be rank of z_j in D_j. In other words, $r_j^{\min}(x)$ (resp. $r_j^{\max}(x)$) is the minimum (resp. maximum) possible rank of x in D_j. We state the following lemma that describes the properties of $r_j^{\min}(x)$ and $r_j^{\max}(x)$:

Lemma 1. *(1) For an element x in the combined summary,*

$$\sum_{j=1}^{t} r_j^{\max}(x) - \sum_{j=1}^{t} r_j^{\min}(x) \le \sum_{j=1}^{t} \varepsilon_j F_1(D_j).$$

(2) For two consecutive elements $x_1 \le x_2$ in the combined summary,

$$\sum_{j=1}^{t} r_j^{\min}(x_2) - \sum_{j=1}^{t} r_j^{\min}(x_1) \le \sum_{j=1}^{t} \varepsilon_j F_1(D_j).$$

Proof. Since $r_j^{\max}(x)$ and $r_j^{\min}(x)$ are the local ranks of two consecutive elements in $\mathcal{S}(\varepsilon_j, D_j)$, we have $r_j^{\max}(x) - r_j^{\min}(x) \le \varepsilon_j F_1(D_j)$. Taking summation over all j, part (1) of the lemma follows. We also note that if x_1 and x_2 are consecutive in the combined summary, $r_j^{\min}(x_1)$ and $r_j^{\min}(x_2)$ are local ranks of either the same element or two consecutive elements of $\mathcal{S}(\varepsilon_j, D_j)$. In either case we have $r_j^{\min}(x_2) - r_j^{\min}(x_1) \le \varepsilon_j F_1(D_j)$. Summing over all j proves part (2) of the lemma. □

Now for each element x in the combined summary, we compute the global minimum rank $r^{\min}(x) = \sum_{j=1}^{t} r_j^{\min}(x)$. Note that all these global ranks can be computed by scanning the combined summary in sorted order. Given a query rank r, we find the smallest element x with $r^{\min}(x) \ge r - \sum_{j=1}^{t} \varepsilon_j F_1(D_j)$. We claim that the actual rank of x in D is in the range $[r - \sum_{j=1}^{t} \varepsilon_j F_1(D_j), r + \sum_{j=1}^{t} \varepsilon_j F_1(D_j)]$. Indeed, we observe that the actual rank of x in set D is in the range $[\sum_{j=1}^{t} r_j^{\min}(x), \sum_{j=1}^{t} r_j^{\max}(x)]$ so we only need to prove that this range is contained by $[r - \sum_{j=1}^{t} \varepsilon_j F_1(D_j), r + \sum_{j=1}^{t} \varepsilon_j F_1(D_j)]$. The left side trivially follows from the choice of x. For the right side, let x' be the largest element in the new summary such that $x' \le x$. By the choice of x, we have $\sum_{j=1}^{t} r_j^{\min}(x') < r - \sum_{j=1}^{t} \varepsilon_j F_1(D_j)$. By Lemma 1 we have $\sum_{j=1}^{t} r_j^{\min}(x) - \sum_{j=1}^{t} r_j^{\min}(x') \le \sum_{j=1}^{t} \varepsilon_j F_1(D_j)$ and $\sum_{j=1}^{t} r_j^{\max}(x) - \sum_{j=1}^{t} r_j^{\min}(x) \le \sum_{j=1}^{t} \varepsilon_j F_1(P_j)$. Summing up these three inequalities yields $\sum_{j=1}^{t} r_j^{\max}(x) \le r + \sum_{j=1}^{t} \varepsilon_j F_1(D_j)$, so the claim follows.

For α-exponentially decomposability, the t data sets have $F_1(D_i) \le \alpha^{i-1} F_1(D_1)$ for $i = 1, \ldots, t$. We choose $c = 1/\sqrt{\alpha} > 1$. The summaries $\mathcal{S}(\varepsilon_1, D_1), \mathcal{S}(\varepsilon_2, D_2), \ldots, \mathcal{S}(\varepsilon_t, D_t)$ have $\varepsilon_i = c^{i-1}\varepsilon$. Therefore we can combine them with error

$$\begin{aligned}\sum_{j=1}^{t} c^{j-1}\varepsilon F_1(D_j) &\le \sum_{j=1}^{t} \left(\frac{\alpha}{\sqrt{\alpha}}\right)^{j-1} \varepsilon F_1(D_1) \\ &= \varepsilon F_1(D_1) \sum_{j=1}^{t} \left(\sqrt{\alpha}\right)^{j-1} \\ &= O(\varepsilon F_1(D_1)) \\ &= O(\varepsilon F_1(D_1 \cup \cdots \cup D_t)).\end{aligned}$$

To combine the t summaries, we notice that we are essentially merging k sorted lists with geometrically decreasing sizes, so we can adapt the algorithm in Section 4.1. The cost of merging the t summaries is therefore $O(s_\varepsilon)$ in internal memory and $O(s_\varepsilon/B)$ I/Os in external memory. The size of combined summary is

$$\sum_{j=1}^{t} \frac{1}{c^{j-1}\varepsilon} = O\left(\frac{1}{\varepsilon}\right) = O(s_\varepsilon).$$

4.3 The Count-Min sketch

Given a multiset D where the items are drawn from a universe $[u] = \{1, \ldots, u\}$. Let $f_D(x)$ be the frequency of x in D. The Count-Min sketch makes use of a 2-universal hash function $h : [u] \to [1/\varepsilon]$ and a collection of $1/\varepsilon$ counters $C[1], \ldots, C[1/\varepsilon]$. Then it computes $C[j] = \sum_{h(x)=j} f_D(x)$ for $j = 1, \ldots, 1/\varepsilon$. A single collection of $1/\varepsilon$ counters achieve a constant success probability for a variety of estimation purposes, and the probability can be boosted to $1 - \delta$ by using $O(\log(1/\delta))$ copies with independent hash functions. Here we only show the decomposability of a single copy; the same result also holds for $O(\log(1/\delta))$ copies.

Given multiple Count-Min sketches with the same h (hence the same number of counters), they can be easily combined by adding up the corresponding counters. So the Count-Min sketch is decomposable. However, for exponentially decomposability we are dealing with t Count-Min sketches with exponentially increasing ε's, hence different hash functions, so they cannot be easily combined. Thus we simply put them together without combining any counters. Although the resulting summary is not a true Count-Min sketch, we argue that it can be used to serve all the purposes a Count-Min is supposed to serve.

More precisely, for t data sets $D_1, \ldots, D_t$ with $F_1(D_i) \leq \alpha^{i-1} F_1(D_1)$, we have t Count-Min sketches $\mathcal{S}(\varepsilon, D_1), \ldots, \mathcal{S}(c^{t-1}\varepsilon, D_t)$. The i-th sketch $\mathcal{S}(c^{j-1}\varepsilon, D_t)$ uses a hash function $h_i : [u] \to [1/c^{j-1}\varepsilon]$. Again we set $c = 1/\sqrt{\alpha}$. Note that the total size of all the sketches is $O(1/\varepsilon + 1/c\varepsilon + 1/c^2\varepsilon + \cdots) = O(1/\varepsilon) = O(s_\varepsilon)$, so we only need to show that the error is the same as what a Count-Min sketch $\mathcal{S}(\varepsilon, D_1 \uplus \cdots \uplus D_t)$ would provide. Below we consider the problem of estimating inner products (join sizes), which has other applications, such as point queries and self-join sizes, as special cases.

Let $\mathbf{f}_i$ denote the frequency vector of D_i, and let $\mathbf{f} = \sum_{i=1}^{t} \mathbf{f}_i$ be the frequency vector of $D = D_1 \uplus \ldots \uplus D_t$. The goal is to estimate inner product $\langle \mathbf{f}, \mathbf{g} \rangle$ where $\mathbf{g}$ is the frequency vector of some other data set. Note that when $\mathbf{g}$ is a standard basic vector (i.e., containing only one "1"), $\langle \mathbf{f}, \mathbf{g} \rangle$ becomes a point query; when $\mathbf{g} = \mathbf{f}$, $\langle \mathbf{f}, \mathbf{g} \rangle$ is the self-join size of $\mathbf{f}$. We distinguish between two cases: (1) $\mathbf{g}$ is given explicitly; and (2) $\mathbf{g}$ is also represented by a summary returned by our index, i.e., a collection of t Count-Min sketches $\mathcal{S}(\varepsilon, G_1), \ldots, \mathcal{S}(c^{t-1}\varepsilon, G_t)$, where $\mathbf{g} = \sum_{i=1}^{t} \mathbf{g}_i$ and $\mathbf{g}_i$ is the frequency vector of G_i. Recall that the Count-Min sketch estimates $\langle \mathbf{f}, \mathbf{g} \rangle$ with an additive error of $\varepsilon F_1(\mathbf{f}) F_1(\mathbf{g})$, and we will show that we can do the same when $\mathbf{f}$ is represented by the collection of t Count-Min sketches.

Inner product with an explicit vector. For a $\mathbf{g}$ given explicitly, we can construct a Count-Min sketch $\mathcal{S}(c^{i-1}\varepsilon, \mathbf{g})$ for $\mathbf{g}$ with hash function h_i, for $i = 1, \ldots, t$. We observe that $\langle \mathbf{f}, \mathbf{g} \rangle$ can be expressed as $\sum_{i=1}^{t} \langle \mathbf{f}_i, \mathbf{g} \rangle$, and $\langle \mathbf{f}_i, \mathbf{g} \rangle$ can be estimated using $\mathcal{S}(c^{i-1}\varepsilon, D_i)$ and $\mathcal{S}(c^{i-1}\varepsilon, \mathbf{g})$ as described in [8] since they use the same hash function. The error is $c^{i-1}\varepsilon ||\mathbf{f}_i||_1 ||\mathbf{g}||_1 \leq (c\alpha)^{i-1}\varepsilon ||\mathbf{f}_1||_1 ||\mathbf{g}||_1$. For $c = 1/\sqrt{\alpha}$, the total error is bounded by

$$\sum_{i=1}^{t} \alpha^{(i-1)/2} \varepsilon ||\mathbf{f}_1||_1 ||\mathbf{g}||_1 = O(\varepsilon ||\mathbf{f}_1||_1 ||\mathbf{g}||_1) = O\left(\varepsilon F_1(\mathbf{f}) F_1(\mathbf{g})\right),$$

as desired.

Inner product with a vector returned by a summary query. Next we consider the case where $\mathbf{g}$ is also represented by a series[4] of t Count-Min sketches $\mathcal{S}(\varepsilon, G_1), \ldots, \mathcal{S}(c^{t-1}\varepsilon, G_t)$ with $F_1(G_i) \geq \alpha^{i-1} F_1(G_1)$. We will show how to estimate $\langle \mathbf{f}, \mathbf{g} \rangle$ using the two series of sketches. This will allow the user to estimate the join size between the results of two queries. Note that this includes the special case of estimating the self-join size of $\mathbf{f}$.

In this case we will inevitably face the problem of pairing two sketches of different sizes. To do so we need more insight into the hash functions used. Suppose $1/\varepsilon$ is a power of 2. Let p be a prime within the range $[u, 2u]$ and a, b be random numbers uniformly chosen from $\{0, \ldots, p-1\}$. If we use the following 2-universal hash functions:

$$h_i(x) = ((ax + b) \bmod p)) \bmod \frac{1}{2^{i-1}\varepsilon}, i = 1, \ldots, \log(1/\varepsilon),$$

then we observe that each bucket of h_{i+1} is partitioned into two buckets of h_i. This means that given a Count-Min sketch $\mathcal{S}(2^{i-1}\varepsilon, D)$ constructed with h_i, one can convert it to a Count-Min sketch $\mathcal{S}(2^{j-1}\varepsilon, D)$ constructed with h_j for any $j \geq i$. Thus two sketches of different sizes can still be used together by reducing the size of the larger one to match that of the smaller one. Of course we will only get the error guarantee of the smaller sketch, but this will not be a problem as we show later.

Now, set $c = 2$ and we have the sketches $\mathcal{S}(2^{i-1}\varepsilon, D_i)$ and $\mathcal{S}(2^{i-1}\varepsilon, G_i)$ with hash function h_i, for $i = 1, \ldots, \log(1/\varepsilon)$. We express $\langle \mathbf{f}, \mathbf{g} \rangle$ as

$$\langle \mathbf{f}, \mathbf{g} \rangle = \left\langle \sum_{i=1}^{t} \mathbf{f}_i, \sum_{i=1}^{t} \mathbf{g}_i \right\rangle = \sum_{i=1}^{t} \langle \mathbf{f}_i, \mathbf{g}_i \rangle + \sum_{i<j} \langle \mathbf{f}_i, \mathbf{g}_j \rangle + \sum_{i<j} \langle \mathbf{g}_i, \mathbf{f}_j \rangle.$$

First, $\langle \mathbf{f}_i, \mathbf{g}_i \rangle$ can be estimated using $\mathcal{S}(2^{i-1}\varepsilon, D_i)$ and $\mathcal{S}(2^{i-1}\varepsilon, G_i)$. The error is at most $2^{i-1}\varepsilon F_1(D_i) F_1(G_i) \leq (2\alpha^2)^{i-1}\varepsilon F_1(D_1) F_1(G_1)$. It follows that $\sum_{i=1}^{t} \langle \mathbf{f}_i, \mathbf{g}_i \rangle$ can be estimated with error $\sum_{i=1}^{t} (2\alpha^2)^{i-1}\varepsilon F_1(D_1) F_1(G_1)$. For $\alpha < 1/\sqrt{2}$, this error is bounded by $O(\varepsilon F_1(D_1) F_1(G_1))$.

For $\langle \mathbf{f}_i, \mathbf{g}_j \rangle$ with $i < j$, we first convert $\mathcal{S}(2^{i-1}\varepsilon, D_i)$ to $\mathcal{S}(2^{j-1}\varepsilon, D_i)$, and do the estimation with $\mathcal{S}(2^{j-1}\varepsilon, G_i)$, which gives us an error of $2^{j-1}\varepsilon F_1(D_i) F_1(G_i)$. Therefore the error of estimating $\sum_{i<j} \langle \mathbf{f}_i, \mathbf{g}_j \rangle$ can be bounded by

$$\begin{aligned}
\sum_{i<j} 2^{j-1}\varepsilon ||\mathbf{f}_i||_1 ||\mathbf{g}_j||_1 &= \sum_{i=1}^{t-1} \varepsilon ||\mathbf{f}_i||_1 \sum_{j=i+1}^{t} 2^{j-1} ||\mathbf{g}_j||_1 \\
&\leq \sum_{i=1}^{t-1} 2^i \varepsilon ||\mathbf{f}_i||_1 \sum_{j=i+1}^{t} (2\alpha)^{j-i-1} ||\mathbf{g}_1||_1 \\
&= \sum_{i=1}^{t-1} 2^i \varepsilon ||\mathbf{f}_i||_1 ||\mathbf{g}_1||_1 \sum_{j=i+1}^{t} (2\alpha)^{j-i-1}.
\end{aligned}$$

For constant $\alpha < 1/2$, we have $\sum_{j=i+1}^{t} (2\alpha)^{j-i-1} = O(1)$, so the error of estimating $\sum_{i<j} \langle \mathbf{f}_i, \mathbf{g}_j \rangle$ is at most

$$\begin{aligned}
O\left(\sum_{i=1}^{t-1} 2^{i-1}\varepsilon ||\mathbf{f}_i||_1 ||\mathbf{g}_1||_1\right) &\leq O\left(\sum_{i=1}^{t-1} (2\alpha^2)^{i-1}\varepsilon ||\mathbf{f}_1||_1 ||\mathbf{g}_1||_1\right) \\
&= O(\varepsilon ||\mathbf{f}_1||_1 ||\mathbf{g}_1||_1).
\end{aligned}$$

We can similarly bound $\sum_{i<j} \langle \mathbf{g}_i, \mathbf{f}_j \rangle = O(\varepsilon ||\mathbf{f}_1||_1 ||\mathbf{g}_1||_1)$.

This proves that Count-Min sketch is α-exponentially decomposable for any constant $0 < \alpha < 1/2$. One technicality is that our data structures only support 1/2-exponentially decomposable summaries as described in Section 3. This is caused by the use of a binary tree $\mathcal{T}$. To get around the problem, we replace the binary tree $\mathcal{T}$ with a ternary tree, so that subtree sizes decrease by a factor of 3 from a level to the one below. Now the left query path may have two nodes on each level in the canonical decomposition of the query range, and so does the right query path. This results in 4 series of sketches for representing $\mathbf{f}$ and $\mathbf{g}$. But this does not affect our analysis by more than a constant factor as argued earlier.

[4]More precisely, $\mathbf{g}$ is represented by two such series: one from the left query path and one from the right query path, and so is $\mathbf{f}$. But we can decompose $\langle \mathbf{f}, \mathbf{g} \rangle$ into 4 subproblems by considering the cross product of these series, where each subproblem involves only a single series of sketches for either $\mathbf{f}$ or $\mathbf{g}$.

Remark. One technical subtlety is that, since we are now making $O(\log^2 N)$ estimations, in order to be able to add up the errors, we need all the estimations to succeed, i.e., stay within the claimed error bounds. To achieve a $1-\delta$ overall success probability, each individual estimation should succeed with probability $1-\delta/\log^2 N$, by the union bound. Thus each Count-Min sketch $\mathcal{S}(c^{i-1}\varepsilon, D_i)$ we use should have size $O((1/c^{i-1}\varepsilon)\log(\frac{\log N}{\delta}))$.

4.4 The AMS sketch and wavelets

Given a multiset D in which the frequency of x is $f_D(x)$, the AMS sketch computes $O((1/\varepsilon^2)\log(1/\delta))$ counters $Y_i = \sum_x h_i(x)f_D(x)$, where each $h_i : [u] \to \{+1,-1\}$ is a uniform 4-wise independent hash function (Dobra and Rusu [25] show that some 3-wise independent hash functions also suffice). The AMS sketch is clearly decomposable. But since it provides an error guarantee depending on $F_2(D)$, it is not exponentially decomposable. Intuitively, the size of a data set could drop by a constant factor without reducing its F_2 significantly. More precisely, for two data sets D_1 and D_2 with $F_1(D_2) \le \alpha F_1(D_1)$ for a constant $\alpha < 1$, $F_2(D_1) - F_2(D_2)$ may be $o(F_2(D_1))$. Thus the AMS sketch can only be used in the baseline solution of Section 2.

Gilbert et al. [12] have shown that an AMS sketch of an appropriate size also incorporates enough information from which we can build a wavelet representation of the underlying data set. Thus, the baseline index of Section 2 can also be used to return a wavelet representation for the data records in the query range.

5. HANDLING UPDATES

If the summary itself supports updates, our indexes also support updates. In particular, the MG summary [23], the GK summary for quantiles [14] support insertions, while the Count-Min sketch and the AMS sketch support both insertions and deletions. The corresponding summary indexes then also support insertions or both insertions and deletions. In this section we briefly describe how we handle updates for the two internal memory structures in Section 2 and 3. The techniques are quite standard [24], so we just sketch the high-level ideas. The external memory indexes also support updates; the details will appear in the full version of the paper.

The baseline structure. We first assume that the structure of the binary tree $\mathcal{T}$ remains unchanged during updates, then we show how to maintain its balance dynamically. We will show how to handle insertions; deletions can be handled similarly, provided that the summary itself supports deletions.

To do an insertion, we first search down the tree $\mathcal{T}$ using the new record's A_q attribute and locate the fat leaf v where the new record should reside. Then we insert it into v. This new insertion affects all the summaries attached to the $O(\log N)$ nodes on the path from the root of $\mathcal{T}$ to v. For each such node u, a summary on all the items stored below u is attached, so we need to insert the new record to the summary as well. Assuming the update cost for a single summary is $O(\mu)$, the total cost of this insertion is $O(\mu \log N)$.

We can maintain the structure of $\mathcal{T}$ using a *weight-balanced* tree and *partial rebuildings* [24]. For any node $u \in \mathcal{T}$, the *weight* of u is defined to be the number of records stored below u. Then we restrict the weight of a node u at height i to vary on the order of $\Theta(2^i s_\varepsilon)$. The fanout of each node of $\mathcal{T}$ may not be 2 any more, but the weight constraint ensures that it is still a constant. After inserting a new record into a leaf v, the weight constraints at the ancestors of v might be violated. Then we find the highest node u where this happens, and simply rebuild the whole subtree rooted at the parent of u. Suppose the parent of u is at height i. We rebuild the subtree level by level. At level j, there are 2^{i-j} summaries we need to build, each on a data set of size $O(s_\varepsilon 2^j)$. So building each summary by simply inserting the records into an initially empty summary takes $O(\mu s_\varepsilon 2^j)$ time. This is $O(\mu s_\varepsilon 2^i)$ in total for level j. Summing over all i levels, the total cost of the rebuilding is $O(\mu s_\varepsilon 2^i \cdot i) = O(\mu s_\varepsilon 2^i \log N)$. After the rebuilding, the weight of u decreases by $\Theta(2^i s_\varepsilon)$, so the cost of the rebuilding can be charged to this weight decrease. Since every insertion increases the weights of $O(\log N)$ nodes by one, the cost of all the rebuildings converts to an $O(\mu s_\varepsilon 2^i \log N / 2^i s_\varepsilon \cdot \log N) = O(\mu \log^2 N)$ cost per insertion amortized.

THEOREM 3. *If the summary can be updated in $O(\mu)$ time, the baseline internal memory structure can be updated in $O(\mu \log^2 N)$ time amortized.*

Remark. In the above rebuilding algorithm, we do not assume any properties of the summary. In fact, for all the summaries considered in this paper, they do not have to be built from scratch for every level. Instead, the summaries at all levels can be constructed more efficiently in $O(\mu s_\varepsilon 2^i)$ time. This will reduce the amortized update cost to $O(\mu \log N)$. Details will appear in the full version of the paper.

The optimal internal memory structure for F_1 based summaries. The update procedure for the optimal internal memory structure of Theorem 1 is almost the same as the baseline solution, except that at each node, we now have $O(\log s_\varepsilon)$ summaries with exponentially decreasing sizes. This adds an $O(\log s_\varepsilon)$ factor to the cost of updating all affected summaries upon each insertion, as well as the partial rebuilding cost. Thus we have:

THEOREM 4. *If the summary can be updated in $O(\mu)$ time, the optimal internal memory structure of Theorem 1 can be updated in $O(\mu \log^2 N \log s_\varepsilon)$ time amortized.*

Remark. The above theorem does not assume any special properties of the summary. Again for all the summaries considered in this paper, the update time can be improved to $O(\mu \log N \log s_\varepsilon)$. Details will appear in the full version of the paper.

6. FUTURE DIRECTIONS

In this paper, we presented some initial positive results on supporting summary queries natively in a database system, that many useful summaries can be extracted with almost the same cost as computing simple aggregates. There are many interesting directions to explore:

1. Our index for the F_2 based summaries does not have the optimal query cost. Can we improve it to optimal? In fact, we can partition the data in terms of F_2 so that the F_2 based summaries are also exponentially decomposable, but we meet some technical difficulties since the resulting tree $\mathcal{T}$ is not balanced.

2. We have only considered the case where there is only one query attribute. In general there could be more than one query attribute and the query range could be any spatial constraint. For example, one could ask the following queries:

   (Q3) Return a summary on the salaries of all employees aged between 20 and 30 with ranks below VP.

   (Q4) Return a summary on the household income distribution for the area within 50 miles from Washington, DC.

   In the most general and challenging case, one could consider any `SELECT-FROM-WHERE` aggregate SQL query and replace the aggregate operator with a `summary` operator.

3. Likewise, the summary could also involve more than one attribute. When the user is interested in the joint distribution of two or more attributes, or the spatial distribution of the query results, a multi-dimensional summary would be very useful. An example is

   (Q5) What is the geographical distribution of households with annual income below $50,000?

   Note how this query serves the complementing purpose of (Q4). To summarize multi-dimensional data, one could consider the multi-dimensional extensions of quantiles and wavelets, as well as geometric summaries such as ε-approximations and various clusterings. The former is useful for multiple relational attributes, while the latter is more suitable for summarizing geometric distributions as in (Q5).

7. REFERENCES

[1] P. Afshani, G. S. Brodal, and N. Zeh. Ordered and unordered top-k range reporting in large data sets. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, 2011.

[2] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In *Advances in Discrete and Computational Geometry*, pages 1–56. American Mathematical Society, 1999.

[3] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. *Journal of Computer and System Sciences*, 64(3):719–747, 2002.

[4] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[5] A. Arasu and G. Manku. Approximate counts and quantiles over sliding windows. In *Proc. ACM Symposium on Principles of Database Systems*, 2004.

[6] K. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla. On synopses for distinct-value estimation under multiset operations. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2007.

[7] G. S. Brodal, B. Gfeller, A. G. Jørgensen, and P. Sanders. Towards optimal range medians. *Theoretical Computer Science*, to appear.

[8] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[9] M. Garofalakis and A. Kumar. Wavelet synopses for general error metrics. *ACM Transactions on Database Systems*, 30(4):888–928, 2005.

[10] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2001.

[11] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proc. ACM Symposium on Theory of Computing*, 2002.

[12] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proc. International Conference on Very Large Data Bases*, 2001.

[13] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[14] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2001.

[15] S. Guha, C. Kim, and K. Shim. XWAVE: Optimal and approximate extended wavelets for streaming data. In *Proc. International Conference on Very Large Data Bases*, 2004.

[16] J. Hellerstein, P. Haas, and H. Wang. Online aggregation. In *Proc. ACM SIGMOD International Conference on Management of Data*, 1997.

[17] C. Jermaine, S. Arumugam, A. Pol, and A. Dobra. Scalable approximate query processing with the dbo engine. *ACM Transactions on Database Systems*, 33(4), Article 23, 2008.

[18] A. Jørgensen and K. Larsen. Range selection and median: Tight cell probe lower bounds and adaptive data structures. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, 2011.

[19] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *Proc. IEEE International Conference on Data Engineering*, 2007.

[20] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *Proc. ACM SIGMOD International Conference on Management of Data*, 1998.

[21] Y. Matias, J. S. Vitter, and M. Wang. Dynamic maintenance of wavelet-based histograms. In *Proc. International Conference on Very Large Data Bases*, 2000.

[22] A. Metwally, D. Agrawal, and A. Abbadi. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Transactions on Database Systems*, 31(3):1095–1133, 2006.

[23] J. Misra and D. Gries. Finding repeated elements. *Science of Computer Programming*, 2:143–152, 1982.

[24] M. H. Overmars. *The Design of Dynamic Data Structures*. Springer-Verlag, LNCS 156, 1983.

[25] F. Rusu and A. Dobra. Pseudo-random number generation for sketch-based estimations. *ACM Transactions on Database Systems*, 32(2), Article 11, 2007.

[26] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

[27] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *Proc. IEEE International Conference on Data Engineering*, 2009.

[28] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16:264–280, 1971.

[29] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proc. SIGMOD International Conference on Management of Data*, 1999.