CoopStore: Optimizing Precomputed Summaries for Aggregation

Edward Gan, Peter Bailis, Moses Charikar
Stanford University
Stanford, CA, USA

ABSTRACT

An emerging class of data systems partition their data and precompute approximate summaries (i.e., sketches and samples) for each segment to reduce query costs. They can then aggregate and combine the segment summaries to estimate results without scanning the raw data. However, given limited storage space each summary introduces approximation errors that affect query accuracy. For instance, systems that use existing mergeable summaries cannot reduce query error below the error of an individual precomputed summary. We introduce CoopStore, a query system that optimizes item frequency and quantile summaries for accuracy when aggregating over multiple segments. Compared to conventional mergeable summaries, CoopStore leverages additional memory available for summary construction and aggregation to derive a more precise combined result. This reduces error by up to 25× over interval aggregations and 4.5× over data cube aggregations on industrial datasets compared to standard summarization methods, with provable worst-case error guarantees.

PVLDB Reference Format:
Edward Gan, Peter Bailis, and Moses Charikar. CoopStore: Optimizing Precomputed Summaries for Aggregation. *PVLDB*, 13(11): xxxx-yyyy, 2020.
DOI: https://doi.org/10.14778/3407790.3407817

1. INTRODUCTION

An emerging class of data systems precompute aggregate summaries over a dataset to reduce query times. These precomputation (AggPre [39]) systems trade off preprocessing time at data ingest to avoid scanning the data at query time. In particular, Druid and similar systems partition datasets into disjoint segments and precompute summaries for each segment [48, 30]. They can then process queries by aggregating results from the segment summaries. Unlike traditional data cube systems [25], the summaries go beyond scalar counts and sums and include data structures that can approximate quantiles and frequent items [16]. As an example, our collaborators at Microsoft often issue queries to estimate 99th percentile request latencies over hours-long time windows. Their Druid-like system precomputes quantile summaries [22, 24] for 5 minute time segments and then combines summaries to estimate quantiles over a longer window, reducing data access and runtime at query time by orders of magnitude [44].

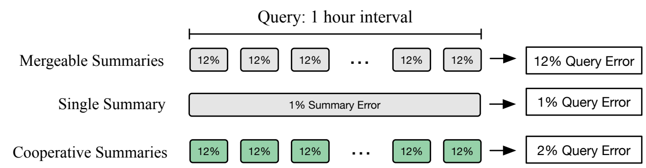


Figure 1: Mergeable summaries preserve accuracy when combined but are less accurate than using a single larger summary. CoopStore closes the gap by using Cooperative summaries optimized for accurate aggregations.

Although querying summaries is more efficient than querying raw data, precomputing summaries also limits query accuracy. Given a total storage budget and multiple data segments, each segment summary in an AggPre system has limited storage space – often <10 kilobytes – and thus limited accuracy [3, 4]. Prior work on *mergeable summaries* introduces summaries that can be combined with no loss in accuracy, and are commonly used in AggPre systems [6, 24, 44]. However, even mergeable summaries have maximum accuracy limited by the accuracy of an individual summary. We illustrate this challenge in Figure 1. Consider a query for the 99th percentile latency from `1:05pm` to `2:05pm`, and suppose we precompute mergeable quantile summaries for 5 minute time segments that individually have 12% error. Calculating quantiles over the full hour requires aggregating results from 12 summaries, and mergeable summaries would maintain 12% error for the final result. This is not ideal: if the same space were instead used to store a single large summary for the entire interval, we would have 12× less error with $\epsilon = 1\%$. On the other hand, using a single large summary restricts the granularity of possible queries.

In this paper we introduce CoopStore, an AggPre query system that uses *Cooperative* item frequency and quantile summaries optimized for accurate aggregations. Unlike mergeable summaries, aggregating results from multiple Cooperative summaries results in lower error than any summary individually. To achieve this CoopStore uses a different resource model than most existing summaries. While merge-

Proceedings of the VLDB Endowment, Vol. 13, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3407790.3407817

able summaries assume the amount of memory available for constructing and aggregating (combining) summaries is the same as that for storage, we have seen in real-world deployments that AggPre systems have orders of magnitude more memory for construction and aggregation.

To keep query times low, AggPre systems can ideally keep summaries cached in memory. However, as data warehouses, AggPre systems must support workloads over many data sets partitioned along multiple dimensions. This means that millions of summaries may have to share limited available memory. Each quantile summary in Druid for instance is configured for 2% error [3] and requires roughly 10 kB of memory to store [4]. During data loading however, Druid can use Hadoop map-reduce jobs that draw on large memory and compute resources for summary construction. Furthermore, at query time engineers at Imply (the company developing Druid) report that standard deployments use query processing jobs with 0.5 GB of memory. CoopStore takes advantage of these additional resources to construct summaries that compensate for the errors in other summaries to reduce final query error.

CoopStore supports queries over intervals and data cubes roll-ups. Interval queries aggregate over one-dimensional contiguous ranges, such as a time window from `1:00pm` to `9:00pm` [9], while data cube queries aggregate over data matching specific dimension values, such as `loc=USA AND type=TCP` [25]. These two query types cover a wide class of common queries and CoopStore can construct summaries optimized for either of the two types. In settings with additional query types, CoopStore and Cooperative summaries can be used alongside existing techniques: one can use the space-efficient Cooperative summaries to improve accuracy on applicable queries and less efficient methods such as online aggregation [28] otherwise.

For both query types, when aggregating over k summaries CoopStore can reduce error by nearly a factor of k for interval aggregations and a factor of $\sqrt{k}$ otherwise, compared with no reduction in error for mergeable summaries. Since modern workloads increasingly require aggregations over hundreds to thousands of summaries, these accuracy improvements are significant. In Figure 2 we describe a set of 33K Top-K item frequency interval queries and 130K quantile interval queries issued to an AggPre system at Microsoft that partitioned data into 5-minute time segments. More than half of the queries span intervals longer than a day and thus aggregate hundreds of summaries. Over half of the data cubes maintained by our collaborators at Microsoft also consisted of more than 10 thousand segments and queries spanning hundreds of cube segments were common.

Interval Queries. For interval queries, CoopStore uses Cooperative summaries that account for the cumulative error over consecutive sequences of summaries, and adjust the error in new summaries to compensate. For instance, if five consecutive item frequency summaries have cumulatively underestimated the true frequency of item x, cooperative summaries can bias the next summary to overestimate x. This keeps the total error for queries spanning k segments small. Hierarchical approximation techniques [9, 42] can also be used here but require additional space and provide worse accuracy in practice.

We prove that our summaries have cumulative error no worse than state of the art randomized summaries [50], and

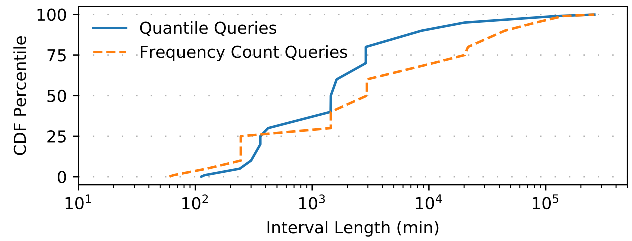


Figure 2: Distribution of user-issued time interval queries to a Druid-like system at Microsoft. More than half of the queries span > 100 five-minute segments.

for frequencies exceed the accuracy of state of the art hierarchical approaches [9]. Empirically, our summaries provide a 4-25× reduction in error on interval queries aggregating multiple summaries compared with existing sketching and summarization techniques.

Multi-dimensional Cube Queries. Data cube queries can aggregate the same summary along different dimensions, so compensating for errors explicitly along a single dimension is insufficient. Instead, for cube workloads CoopStore uses Cooperative summaries that consist of weighted random samples (PPS samples [15]) optimized specifically for data cube workloads. CoopStore exploits the fact that data cubes often have dimensions with skewed value distributions: some values or combination of values occur far more frequently than others. Then, CoopStore optimizes the allocation of storage space and statistical bias among the summaries to minimize average query error. Empirically, these optimizations yield an up to 4.5× reduction in average error over data cube queries compared with standard data cube summarization techniques.

In summary, we make the following contributions:

1. We introduce CoopStore, an approximate AggPre system that provides improved query accuracy for item frequency and quantile aggregations over multiple summaries by taking advantage of additional memory resources at data ingest and query time.

2. We develop novel Cooperative summaries for interval queries with improved worst-case error bounds for large interval aggregations.

3. We develop Cooperative summaries for data cube queries with optimized space usage and bias to minimize average error under cube aggregations.

The remainder of the paper proceeds as follows. In Section 2 we present CoopStore and its query model. In Section 3 we describe Cooperative summaries optimized for intervals. In Section 4 we describe Cooperative summaries optimized for data cubes. In Section 5 we evaluate CoopStore accuracy and runtime. We describe related work in Section 6 and conclude in Section 7.

2. SYSTEM DESIGN

In this section we describe CoopStore's system design. We discuss the types of queries supported, how summaries are constructed for different query types, and how summaries

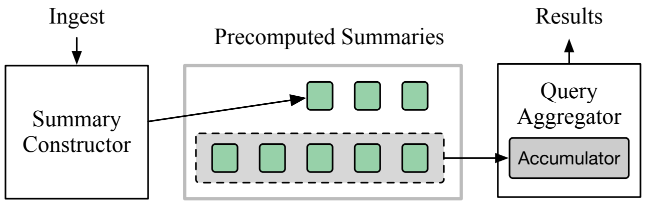


Figure 3: CoopStore precomputes summaries at ingest (Section 2.2) optimized to minimize error under aggregations. At query time, results from multiple summaries are combined using a precise accumulator (Section 2.3).

are aggregated to provide accurate query results. We outline the system components in Figure 3.

2.1 Queries

Consider data records $\rho = (x, t, d_1, \ldots, d_{m_d})$ where x is either a categorical or ordinal value of interest (i.e. ip address, latency), t is an ordered dimension for interval queries (i.e. timestamp), and the d_j are categorical dimensions (i.e. location). A CoopStore query $g_{\mathcal{Q}}(x)$ specifies an aggregation of records $\mathcal{Q}$ and a function g to estimate for the value x. $\mathcal{Q}$ defines an *aggregation* with a selection condition: either a one-dimensional interval or a multi-dimensional cube query [9, 25].

Definition 1. An interval aggregation specifies $\mathcal{Q}^{(\text{interval})} = \{\rho : T_0 \leq t < T_1\}$ for T_0, T_1 aligned at a time-resolution T_G ($T_0, T_1 \bmod T_G = 0$) and maximum length $T_1 - T_0 \leq k_T \cdot T_G$.

Definition 2. A data cube aggregation specifies $\mathcal{Q}^{(\text{cube})} = \{\rho : d_{i_1} = v_{i_1} \wedge \ldots \wedge d_{i_k} = v_{i_k}\}$ for $d_{i_1}, \ldots, d_{i_k}$ a subset of the dimensions to condition on.

The *query function* g is either an item frequency f or rank r [17, 33]. An item frequency $f(x)$ is the total count of records with value x while a rank $r(x)$ is the total count of records with values less than or equal to x. We use g generically to denote either frequencies or ranks.

$$f_{\mathcal{Q}}(x) = \sum_{\rho_i \in \mathcal{Q}} 1_{x_i = x} \qquad r_{\mathcal{Q}}(x) = \sum_{\rho_i \in \mathcal{Q}} 1_{x_i \leq x}. \qquad (1)$$

Using these primitives, CoopStore can also return estimates for quantiles and Top K / Heavy Hitters queries, which we will discuss in more detail in Section 2.3.

2.2 Data Ingest

Before CoopStore can ingest data, users specify whether they want to support interval or data cube aggregations, and whether they are interested in frequency or rank query functions. Users also specify total space constraints and workload parameters. A dataset can be loaded multiple times to support different combinations of the above. Like Druid [48], segment summaries in CoopStore are immutable once created so data updates can be done by re-ingesting data for any segments with updated data.

CoopStore then splits the data records into atomic segments $\mathcal{D}$. These segments form a disjoint partitioning of a dataset, and are chosen so that any aggregation can be expressed as a union of segments. For interval aggregations users specify a time resolution T_G and a maximum length k_T, defining segments $\mathcal{D}_i = \{\rho : i \cdot T_G \leq t < (i+1) \cdot T_G\}$. For cube aggregations the partitions are defined by grouping by all m_d of the dimensions $\mathcal{D}_{\vec{v}} = \{\rho : d_1 = v_1 \wedge \cdots \wedge d_{m_d} = v_{m_d}\}$. Defining finer partitionings into more segments allows for more flexible queries but can degrade query accuracy as seen in Figure 1 and later in our evaluation in Figure 8. CoopStore can alleviate this degradation so in practice users should define partitions based on the smallest resolution they expect to be relevant for querying.

Once the dataset is partitioned we can represent the records in each segment $\mathcal{D}$ as mappings from item values x_j to counts δ_j, and for each segment CoopStore constructs a Cooperative summary S consisting of s value, count mappings

$$\mathcal{D} = \{x_j \mapsto \delta_j : x_j \in \mathcal{D}\}$$
$$S = \{x_{j_1} \mapsto \gamma_{j_1}, \ldots, x_{j_s} \mapsto \gamma_{j_s}\}.$$

This is similar to other counter based summaries [35, 17] and weighted sampling summaries [50]. Unlike tabular sketches such as the Count-Min Sketch [19] Cooperative summaries include the item values x. We assume we have enough memory and compute to generate S, making our construction routines similar to coreset construction [41]. More details on how the values x and counts γ are chosen for each summary are given in Section 3.1 for interval aggregations and Section 4.1 for cube aggregations. Interval summaries can be constructed sequentially in one pass over the data segments while cube summaries require two passes: one to optimize summary parameters and one to construct each summary.

2.3 Query Processing

After the summaries have been constructed, the CoopStore query processor can return query estimates $\hat{g}_{\mathcal{Q}}(x)$ for different aggregations $\mathcal{Q}$ by using the summaries S_i as proxies for the segments $\mathcal{D}_i$. Then, using g to denote a generic query function, we can derive frequency or rank estimates over a query aggregation $\mathcal{Q}$ by adding up the estimates for the segment summaries.

$$f_S(x) \coloneqq \sum_{x_j \in S} \gamma_j \cdot 1_{x_j = x} \qquad r_S(x) \coloneqq \sum_{x_j \in S} \gamma_j \cdot 1_{x_j \leq x}$$

$$\hat{g}_{\mathcal{Q}}(x) = \sum_{S_i \in \mathcal{Q}} g_{S_i}(x) \qquad (2)$$

For rank or frequency estimates for a specific item $\hat{g}_{\mathcal{Q}}(x)$ a query processor can accumulate estimates using Equation 2. To support queries for ranks and counts of potentially unknown items, for instance in quantile and top-k queries, CoopStore accumulates items and counts from summaries into an accumulator A which is a map tracking all items and their cumulative counts. Note that unlike systems that use mergeable summaries, the accumulator A can grow to be larger than any individual summary. The accumulator A can then be queried for quantiles or top item frequencies. Given sufficient memory, A can track items and counts precisely incurring no additional error beyond the error inherent in the cumulative items and counts of the summaries.

When memory is constrained we instead let A be a standard but very large stream summary of the proxy values and counts stored in $S_1, \ldots, S_k$. We specifically use a Space Saving sketch [35] for heavy hitters and a PPS (`VarOpt` [15]) sample for quantiles. In practice the space s_A available to A is orders of magnitude greater than the space s available to any precomputed summary, i.e. 50,000× in the deployments

Table 1: Summary Error ignoring constants combining k summaries. The Cooperative summaries used by CoopStore have reduced errors for large k.

Summary	$\epsilon_{\mathcal{Q}}$	Total Space
Coop. Cube (PPS)	$1/(s\sqrt{k})$	sk
Coop. Interval Freq.	$\log k_T/(sk)$	sk
Coop. Interval Quant.	$\sqrt{k_T}/(sk)$	sk
Mergeable [6]	$1/s$	sk
Uniform Sample	$1/\sqrt{sk}$	sk
Hierarchical [9, 42]	$\log k/(sk)$	$sk \log k_T$

at Imply described in Section 1.

Our prototype implementation of CoopStore is a single-node system, but can be extended to distributed settings following a standard design where distributed query processors aggregate results into partial accumulators and reducer(s) combine the partial accumulators (See [10] for an example).

2.4 Error Model

Consider the absolute (i.e. unscaled) error

$$\varepsilon_{\mathcal{Q}}(x) = g_{\mathcal{Q}}(x) - \hat{g}_{\mathcal{Q}}(x) = \sum_{\mathcal{D}_i \in \mathcal{Q}} (g_{\mathcal{D}_i}(x) - g_{S_i}(x)) \quad (3)$$

which is the difference between the true and estimated item frequency counts or ranks. Throughout the paper, we denote the absolute count or rank error with ε, but compare final query accuracy based on the normalized (scaled) error $\epsilon = \varepsilon/|\mathcal{Q}|$ [6] where $|\mathcal{Q}|$ is the number of items encompassed in the query $|\mathcal{Q}| = \sum_{\rho_i \in \mathcal{Q}} 1$. Unless otherwise stated we will use 'error' to refer to the normalized error. Limited-memory approximate accumulators A would introduce additional error $\varepsilon^{(\mathrm{A})}$ (zero for precise accumulators) in approximating the proxy item counts in the summaries S, yielding an absolute error of:

$$\varepsilon_{\mathcal{Q}}^{(\mathrm{A})}(x) \leq |\varepsilon_{\mathcal{Q}}(x)| + |\varepsilon^{(\mathrm{A})}(x)|. \quad (4)$$

Furthermore we are interested in systems that provide error bounds over all values of x, so we consider the worst case error $\varepsilon_{\mathcal{Q}}^{(\mathrm{A})} := \max_x |\varepsilon_{\mathcal{Q}}^{(\mathrm{A})}(x)|$. A bound on the maximum error over all x also bounds the error of any quantile or heavy hitter frequency estimate derived from the raw estimates $\hat{g}$.

To analyze the error, consider an aggregation $\mathcal{Q}$ accumulating k segments, each with the same number of records $n = |\mathcal{D}| = \sum_{x_i \in \mathcal{D}} \delta_i$ and represented using summaries of size s. Also, suppose that the accumulator A has size $s_A \gg s$, so that $\varepsilon^{(\mathrm{A})} = 0$. Suppressing logarithmic factors, state of the art frequency and quantile summaries have absolute error $O(n/s)$ [31, 17, 6, 41]. Different summarization techniques yield different errors as the size of the aggregation k grows. CoopStore can reduce error significantly for large k. We summarize bounds on the normalized error in Table 1.

Merging mergeable summaries [6] preserves normalized error so we have

$$\epsilon_{\mathcal{Q}}^{(\mathrm{merge})} \leq O(1/s). \quad (5)$$

Instead of merging, using an exact accumulator applied to the estimates of standard summaries gives us $\varepsilon_{\mathcal{Q}}^{(\mathrm{A})\mathrm{naive}} \leq \sum_{\mathcal{D}_i \in \mathcal{Q}} |O(n/s)|$ so we also have

$$\epsilon_{\mathcal{Q}}^{(\mathrm{A})\mathrm{naive}} \leq O(1/s). \quad (6)$$

Table 2: Notation Reference

ϵ	Normalized error	ε	Absolute error
$\mathcal{D}$	Data Segment	S	Data Summary
$\lvert\mathcal{D}\rvert$	# items in segment	s	Summary space
$f(x)$	Item Freq.	$r(x)$	Item rank
$g(x)$	Freq. or Rank	Pre_t	Prefix interval
k_T	Max Interval length	k	Segments in query

However, CoopStore is able to achieve lower query error by reducing the sum of errors from summaries in Equation 3. By using independent, unbiased, weighted random samples – specifically PPS summaries in Section 4.1 – sums of random errors centered around zero will concentrate to zero, and one can use Hoeffding's inequality to show that with high probability and ignoring log terms $\sum_{\mathcal{D}_i \in \mathcal{Q}} \varepsilon_{\mathcal{D}_i}(x) \leq O(\sqrt{k}n/s)$ so

$$\epsilon_{\mathcal{Q}}^{(\mathrm{A})\mathrm{PPS}} \leq O\left(\frac{1}{\sqrt{ks}}\right). \quad (7)$$

This already is lower than the error for mergeable summaries in Equation 5 for $k \gg 1$.

In practice, Cooperative summaries (Section 3) achieve even better error than PPS summaries. Cooperative summaries for data cubes use PPS and introduce further size and bias optimizations, while Cooperative interval summaries use more sophisticated specialized algorithms.

We can prove that Cooperative quantile summaries over intervals satisfy worst-case bounds similar to standard PPS summaries but have much lower error on real-world datasets. Cooperative frequency summaries over intervals satisfy $\max_x |\varepsilon_{\mathcal{Q}}^{(A)\mathtt{CoopFreq}}(x)| \leq O(n \log k_T/s)$

$$\epsilon_{\mathcal{Q}}^{(A)\mathtt{CoopFreq}} \leq O\left(\frac{\min(\log k_T, k)}{ks}\right) \quad (8)$$

where k_T is the maximum length of an interval, which is much stronger than the guarantees provided by standard PPS summaries. See Section 3.3 for more details and proof sketches.

Hierarchical estimation is a common solution for interval (range) queries [9, 19] and show up in differential privacy as well [18]. We will describe an instance of these methods to illustrate their error scaling. A dyadic (base 2) hierarchy stores summaries of size $s \cdot 2^h$ for $h = 1 \ldots \log k_T$ to track segments of different lengths. They can thus estimate intervals of length k with error $\varepsilon_{\mathcal{Q}}^{(\mathrm{A})\mathrm{Hier}} \leq O(n \log k/s)$, similar to our cooperative frequency sketches. However, they incur an additional $\log k_T$ factor in space usage to maintain their multiple levels of summaries and provide worse error empirically than Cooperative summaries.

3. COOPERATIVE INTERVAL SUMMARIES

In this section we describe Cooperative summaries for interval queries. These summaries achieve high query accuracy when aggregated by compensating for accumulated errors over sequences of summaries.

3.1 Interval Summary Construction

Given space for s counters $(x_i \mapsto \gamma_i)$, Cooperative summaries must accurately represent a single segment of data. To match state of the art summary error on a single segment

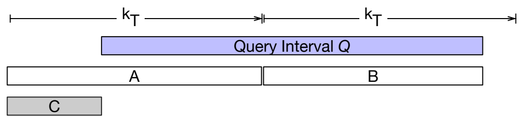


Figure 4: Any contiguous interval can be expressed as a linear combination of aligned intervals Pre_t. In this example, Q is expressed as $A \cup B \setminus C$

$\mathcal{D}$ we want

$$\max_x |\hat{g}_S(x) - g_{\mathcal{D}}(x)| \le r|\mathcal{D}|/s \tag{9}$$

for an accuracy parameter $r \ge 1$. r is an adjustable hyper-parameter: larger r allows for larger errors on a single segment but allow Cooperative summaries to better control error accumulation over longer intervals (See Theorem 1 in Section 3.3). However, there are multiple ways to choose the items to store in S that would satisfy Equation 9.

Within these constraints, CoopStore can choose x_i, γ_i to minimize the total error for queries that aggregate multiple summaries. CoopStore explicitly minimizes the error over intervals with fixed start points every k_T segments, where k_T is the maximum supported query interval length. We call these aggregation intervals "prefix" intervals Pre_t, a modification of standard prefix-sum ranges [29].

$$\mathrm{Pre}_t = \{\mathcal{D}_{k_T\lfloor t/k_T \rfloor}, \dots, \mathcal{D}_{k_T\lfloor t/k_T \rfloor + t \bmod k_T}\}. \tag{10}$$

Figure 4 illustrates how any consecutive interval of up to k_T segments can be represented as an additive combination of up to 3 prefix intervals. As long as prefix intervals have bounded error $\varepsilon_{\mathrm{Pre}_t} = g_{\mathrm{Pre}_t} - \hat{g}_{\mathrm{Pre}_t}$, any contiguous interval of segments up to length k_T has error at most 3ε. Thus, CoopStore constructs Cooperative summaries for intervals incrementally, tracking the cumulative error over prefix intervals $\varepsilon_{\mathrm{Pre}_t}(x)$ in order to construct summaries that minimize this error. However, at query time, the summaries can be aggregated without consideration for the prefix intervals.

Example. Consider constructing Cooperative summaries for frequency queries over intervals with accuracy parameter $r = 1.1$. Since CoopStore constructs interval summaries incrementally, suppose CoopStore has already constructed summaries for three successive time segments $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3$ since the last prefix start point and is now constructing a summary S_4 with size $s = 4$ for time segment $\mathcal{D}_4$ with 1000 items. To accurately represent $\mathcal{D}_4$, S_4 must include any item in $\mathcal{D}_4$ that occurs with frequency at least $r \cdot |\mathcal{D}_4|/4 = 275$, in our example assume there are two such items $\mathcal{A}$ which occurs 500 times and $\mathcal{B}$ which occurs 300 times.

Our method for constructing S_4 will store the true counts for $\mathcal{A}, \mathcal{B}$ and use the remaining space for two counters to store items which have been *underrepresented* cumulatively in the summaries from the current prefix interval S_1, S_2, S_3 and compensate for the discrepancy Δ. If $\mathcal{F}$ and $\mathcal{H}$ are the two items most severely underrepresented then S_4 will store the current counts for $\mathcal{F}, \mathcal{H}$, which occur 10 and 5 times in $\mathcal{D}_4$, adjusted up to the error tolerance $r \cdot |\mathcal{D}_4|/4$ to reduce the cumulative summarization error for those items. In our example suppose this cumulative discrepancy $\Delta > 275$ for $\mathcal{F}, \mathcal{H}$. S_4 is then $\{\mathcal{A} \mapsto 500, \mathcal{B} \mapsto 300, \mathcal{F} \mapsto 285, \mathcal{H} \mapsto 280\}$

3.2 Interval summary details

The details of the summary construction algorithm differ for frequencies and ranks and we present the pseudocode for their construction below.

Algorithm 1 Cooperative Frequencies on Intervals

function $\textsc{CoopFreq}(\mathcal{D}_t, s)$
  $h \leftarrow |\mathcal{D}_t|/s$
  $\varepsilon_{\mathrm{Pre}_t}(x) \leftarrow \varepsilon_{\mathrm{Pre}_{t-1}}(x) + f_{\mathcal{D}_t}(x)$
  $S_t \leftarrow \{x \mapsto f_{\mathcal{D}_t}(x) : f_{\mathcal{D}_t}(x) \ge h\}$ ▷ Heavy hitters
  while $|S_t| < s$ **do** ▷ Correct Accumulated Errors
    $x_m \leftarrow \arg\max_{x \in \mathrm{Pre}_t \setminus S_t} (\varepsilon_{\mathrm{Pre}_t}(x))$
    $\delta_m \leftarrow \min(r \cdot h, \varepsilon_{\mathrm{Pre}_t}(x))$
    $S_t \leftarrow S_t \cup \{x_m \mapsto \delta_m\}$
    $\varepsilon_{\mathrm{Pre}_t}(x) \leftarrow \varepsilon_{\mathrm{Pre}_t}(x) - \delta_m \cdot 1_{x=x_s}$
  return S_t

In Algorithm 1 we present the pseudocode for constructing a cooperative summary of size s for frequency estimates on a data segment $\mathcal{D}_t$. To satisfy Equation 9 and accurately represent $\mathcal{D}_t$, we store the true count for any segment-local heavy hitter items in $\mathcal{D}_t$ that occur with count greater than $|\mathcal{D}_t|/s$. The remaining space in the summary is allocated to storing adjusted counts for the x with the highest cumulative undercount $\varepsilon_{\mathrm{Pre}_t}(x)$ to correct for accumulated approximation error. The adjusted count is the smaller of $r|\mathcal{D}|/s$ and $\varepsilon_{\mathrm{Pre}_t}(x)$. This ensures Equation 9 is satisfied and also keeps $\varepsilon_{\mathrm{Pre}_t}(x)$ positive, a useful invariant for proofs later. Larger r allow the algorithm trade off higher local error for less error accumulation across summaries.

Algorithm 2 Cooperative Quantiles on Intervals

function $\textsc{CoopQuant}(\mathcal{D}_t, s)$
  $h \leftarrow |\mathcal{D}_t|/s$; $S_t \leftarrow \{\}$
  $\varepsilon_{\mathrm{Pre}_t}(x) \leftarrow \varepsilon_{\mathrm{Pre}_{t-1}}(x) + r_{\mathcal{D}_t}(x)$
  $\mathcal{D}_{t1}, \dots, \mathcal{D}_{ts} \leftarrow \textsc{Partition}(\mathcal{D}_t, s)$ ▷ Sorted Chunks
  for $i \in 1 \dots s$ **do**
    $L(z) := \sum_{y \in U} \phi(\varepsilon_{\mathrm{Pre}_t}(y))$
    $x_s \leftarrow \arg\min_{z \in \mathcal{D}_{ti}} L(z)$ ▷ Minimize Loss
    $S_t \leftarrow S_t \cup \{x_s \mapsto h\}$
    $\varepsilon_{\mathrm{Pre}_t}(x) \leftarrow \varepsilon_{\mathrm{Pre}_t}(x) - h \cdot 1_{x \ge x_s}$
  return S_t

In Algorithm 2 we present pseudocode for constructing a cooperative summary of size s for rank estimates on a data segment $\mathcal{D}_t$. To satisfy Equation 9 and accurately represent $\mathcal{D}_t$, we sort the values in $\mathcal{D}$ and partition the sorted values into s equally sized chunks. Then `CoopQuant` selects one value in each chunk to include in S_t as a representative with proxy count $|\mathcal{D}|/s$. This ensures that any rank can be estimated using S_t with error at most $|\mathcal{D}|/s$. Within each chunk, we store the item that minimizes a total loss $L = \sum_{x \in U} \phi(\varepsilon_{\mathrm{Pre}_t}(x))$ with $\phi(\epsilon) = \cosh(\alpha\epsilon)$, $\alpha = s/(\sqrt{k_T} n_{\max})$, $n_{\max} = \max_t |\mathcal{D}_t|$ the maximum size of a data segment, and k_T the maximum interval length. $\cosh(x) = \frac{1}{2}(e^x + e^{-x})$ is used in discrepancy theory [46] to exponentially penalize both large positive and large negative errors, so L serves as a proxy for the L_∞ maximum error. Note that we need to bound $n_{\max}$ to set α for this algorithm, though in practice accuracy changes very little depending on $n_{\max}$

3.3 Interval Query Error

CoopFreq and CoopQuant both provide estimates with local absolute error $\varepsilon_{\mathcal{D}}(x) \leq r|\mathcal{D}|/s$ for a single segment $\mathcal{D}$, and minimize the cumulative absolute error over $\varepsilon_{\mathrm{Pre}_t}(x)$ prefix intervals (and thus general intervals). In this section we analyze how $\varepsilon_{\mathrm{Pre}_t}(x)$ grows with t. This allows us to establish the bounds on the query error when aggregating any sequence of k_T Cooperative summaries (Used in Section 2.4).

The general strategy will be to define a loss L_t which is a function of the absolute errors $\varepsilon_{\mathrm{Pre}_t}(x)$ parameterized by a cost function ϕ

$$L_t := \sum_{x \in U} \phi\left(\varepsilon_{\mathrm{Pre}_t}(x)\right) \tag{11}$$

where U is the universe of observed values $x \in |\mathrm{Pre}_t|$. We can bound the growth of L_t when CoopQuant and CoopFreq are used to construct sequences of summaries. Then, we can relate L_t and $\max_x |\varepsilon_{\mathrm{Pre}_t}(x)|$ to bound the latter. Omitted proofs can be found in an associated technical report [23].

3.3.1 CoopFreq *Error*

For frequency summaries, we use the cost function $\phi(x) = \exp(\alpha x)$ for a parameter α. Since Algorithm 1 always produces underestimates for counts in prefix intervals, the error $\varepsilon_{\mathrm{Pre}_t}(x)$ is always positive so we can minimize L_t as a proxy for the maximum error. Lemma 1 bounds how much L_t can increase with t.

LEMMA 1. *When* ***CoopFreq*** *constructs a summary with size s for $\mathcal{D}_t$ the loss satisfies*

$$L_t \leq L_{t-1} + \alpha r |\mathcal{D}_t|$$

for $\phi(x) = \exp(\alpha x)$ as long as $0 < \alpha \leq 2\frac{s}{|\mathcal{D}_t|}\frac{r-1}{r^2}$.

Given this lemma, we can bound the growth of L_t and relate that to the cumulative error:

THEOREM 1. ***CoopFreq*** *maintains*

$$\max_{x \in U} |\varepsilon_{Pre_t}(x)| \leq \frac{1}{\alpha} \ln\left(1 + \alpha r \sum_{i=1}^{t} |\mathcal{D}_i|\right)$$

where $\alpha = 2\frac{s}{\max_i |\mathcal{D}_i|}\frac{r-1}{r^2}$.

To illustrate the asymptotic behavior we can apply Theorem 1 with $r = \frac{3}{2}$ and consistent segment weights $n = |\mathcal{D}_i|$ to see in Corollary 1 that the absolute error grows logarithmically with the number of segments k in the interval.

COROLLARY 1. *For $r = \frac{3}{2}$ and $|\mathcal{D}_i| = n$,* ***CoopFreq*** *maintains*

$$\max_{x \in U} |\varepsilon_k(x)| \leq \frac{9}{4}\frac{n}{s} \ln\left(1 + \frac{2}{3}nk\right)$$

In fact this result is close to optimal: an adversary generating incoming data can guarantee at least $\Omega(\log k)$ error accumulation by generating data containing items the summaries have undercounted the most so far.

3.3.2 CoopQuant *Error*

For rank queries we use the cost function $\phi(x) = \cosh \alpha x$, inspired by previous work in discrepancy theory [46]. Since $\cosh z = \frac{1}{2}(\exp(z) + \exp(-z))$ this exponentially penalizes both under and over-estimates symmetrically, and is thus a smooth proxy for the maximum absolute error. As with CoopFreq, Lemma 2 bounds the growth of L_t.

LEMMA 2. *When* ***CoopQuant*** *constructs a summary with size s for $\mathcal{D}_t$ the loss function satisfies*

$$L_t \leq L_{t-1} \exp \alpha^2 (|\mathcal{D}_t|/s)^2/2$$

for $\phi(x) = \cosh(\alpha x)$

As with frequency errors, we can then bound the growth of L_t and relate that to the cumulative error:

THEOREM 2. ***CoopQuant*** *maintains*

$$\max_{x \in U} |\varepsilon_{Pre_t}(x)| \leq \frac{1 + 2\ln(2|U|)}{2s} \sqrt{\sum_{i}^{t} |\mathcal{D}|_t^2}$$

with $\phi(x) = \cosh(\alpha x)$ and $\alpha = s\left(\sum_{i=0}^{t} |\mathcal{D}_i|^2\right)^{-1/2}$.

This can be instantiated for data segments with constant total weight in Corrollary 2, which shows that CoopQuant has absolute error $O(\sqrt{k}/s)$.

COROLLARY 2. *For $|\mathcal{D}_t| = n$ constant and $\phi(x) = \cosh(\alpha x)$ with $\alpha = \frac{s}{n\sqrt{k}}$,* ***CoopQuant*** *maintains*

$$\max_{i \in U} |\varepsilon_{Pre_k}(i)| \leq \frac{n}{2s}\left(\sqrt{k} + 2\ln(2|U|)\right)$$

4. COOPERATIVE CUBE SUMMARIES

For cube queries, CoopStore uses Cooperative summaries that consists of weighted probability proportional to size samples (PPS) [15, 47] with an optimized allocation of space and bias between the summaries to improve average query accuracy across data cube aggregations.

4.1 PPS Summaries

A PPS summary is a weighted random sample that includes items with probability proportional to their size or total count in a data segment $\mathcal{D}$ [15, 47, 50]. Values x_i with true occurence count $\mathcal{D}(x_i) = \delta_i$ are sampled for inclusion in the summary S according to Equation 12

$$\Pr[x_i \in S] = \min(1, \delta_i/h) \tag{12}$$

$$S(x_i) = \begin{cases} h & \delta_i \leq h \\ \delta_i & \delta_i > h \end{cases} \tag{13}$$

for an accuracy parameter h. Heavy hitters that occur more than h times are always sampled with their true count, while those with count $0 \leq \delta_i \leq h$ are either included with a proxy count of h or excluded from the summary. Thus, $S(x_i)$ is an unbiased estimate for δ_i with maximum local error of h. Following details in [15] one can set h using a procedure we denote CalcT (Algorithm 4, Stream-τ in [15]) that ensures the summary will have size at most s. Cooperative summaries then store items to ensure either rank or frequency estimates have absolute segment error bounded by h.

4.2 Cube Summary Construction

Cooperative summaries for data cubes are constructed for an entire cube in batch with a total space budget S_T. This opens up the opportunity for optimizations across the entire collection of summaries.

In most multi-dimensional data cubes some queries and dimension values will be much rarer than others. This makes it wasteful to optimize for worst-case error: even the rarest data segment would require the same error and space as more representative segments of the cube. Thus, Cooperative data cube summaries make use of limited space by optimizing the allocation of space and bias between summaries to minimize the average error of queries sampled from a probabilistic workload W specified by the user. We show in Section 5.3.1 that the workload does not have to be perfectly specified to achieve accuracy improvements.

Example. Consider item frequency queries for items x in a data cube with four segments defined by two binary dimensions $d_1, d_2 \in \{0, 1\}$. In this example, the distribution of dimension value pairs (d_1, d_2) among the data is 70% $(0, 0)$, 20% $(0, 1)$, 7% $(1, 0)$ and 3% $(1, 1)$. A query $\mathcal{Q}_1$ for item frequencies over the entire cube would consist primarily of items from the segments for $(0, 0)$ and $(0, 1)$, so to minimize the error for $\mathcal{Q}_1$ one would allocate more space to the summaries corresponding to $(0, 0)$ and $(0, 1)$ than the others. However, another query $\mathcal{Q}_2$ for item frequencies with $(d_1, d_2) = (1, 1)$ would benefit solely from the space to the summary for that segment. Given an expected workload of queries CoopStore allocates space between Cooperative summaries to balance these competing concerns: for a workload where $\mathcal{Q}_1$ is very likely an optimal summary space allocation could be 40% $(0, 0)$, 30% $(0, 1)$, 20% $(1, 0)$ and 10% $(1, 1)$.

4.3 Minimizing Average Error

Consider the error incurred by combining summaries over a query $\mathcal{Q} = \mathcal{D}_1, \ldots, \mathcal{D}_k$, where the segment summaries S_i have size s_i and represent segments with total count $|\mathcal{D}_i| = n_i$. Then, based on Equation 12, the normalized error $\epsilon_{\mathcal{Q}}(x)$ is a random variable that depends on the items selected for inclusion in the PPS summaries. We will bound the mean squared error $E[\epsilon(x)^2]$.

For a single segment $\mathcal{D}_i$, the PPS summary is unbiased and returns both frequency and rank estimates that lie within a possible range of length h. Thus, the absolute error satisfies $E[\varepsilon_{\mathcal{D}_i}(x)] = 0$ and $E[\varepsilon_{\mathcal{D}_i}(x)^2] \leq \frac{1}{4}h^2 \leq \frac{1}{4}n_i^2/s_i^2$, and since the summaries S_i are independent:

$$E[\varepsilon_{\mathcal{Q}}^2] \leq \frac{1}{4} \sum_{\mathcal{D}_i \in \mathcal{Q}} \left(\frac{n_i}{s_i}\right)^2.$$

We consider a workload W as a distribution over possible queries $\mathcal{Q}_i$ where $\Pr[\mathcal{Q}_i \sim W] = q_i$. This is based off of the workloads in STRAT [13], though STRAT targets only count and sum queries using simple uniform samples. To limit worst-case accuracy, we can optionally impose a minimium size for each segment summary $s_{\min}$ so that the maximum relative error for any query is $\epsilon \leq \frac{1}{s_{\min}}$.

Space Allocation. Now, we minimize the mean squared error for queries drawn from a workload $\mathcal{Q}_z \sim W$ where $\Pr[\mathcal{Q}_z] = q_z$. Let $|\mathcal{Q}_z| = \sum_{\mathcal{D}_i \in \mathcal{Q}_z} |\mathcal{D}_i|$.

$$E_{\mathcal{Q}_z \sim W}\left[\epsilon_{\mathcal{Q}}^2\right] \leq \frac{1}{4} \sum_{\mathcal{D}_i \in \mathcal{D}} \frac{n_i^2}{s_i^2} \left(\sum_{z | \mathcal{D}_i \in \mathcal{Q}_z} q_z |\mathcal{Q}_z|^{-2} \right) \quad (14)$$

We can solve for the s_i that minimize the RHS of Equation 14 under the total space constraint that $\sum_i s_i = S_T$ using Lagrange multipliers. The optimal s_i are $s_i \propto \alpha_i^{1/3}$ where

$$\alpha_i = n_i^2 \sum_{z | \mathcal{D}_i \in \mathcal{Q}_z} q_z |\mathcal{Q}_z|^{-2} \quad (15)$$

Since we can compute α_i given W, this gives us a closed form expression for an allocation of storage space.

Bias and Variance. When estimating item frequencies, we can further reduce error by tuning the bias of our Cooperative summaries to reduce their variance. Though this does not generalize to quantile queries, the improvements in accuracy for frequency queries can be substantial, and we have not seen other systems optimize for bias across a collection of summaries.

For example, consider a segment $\mathcal{D}$ with $n > 4$ unique items that each only occur once. If we summarize the data with an empty summary, estimating 0 for the count of each item, we introduce a fixed bias of 1 but have a deterministic estimator with no variance. This substantially reduces the error compared to an unbiased PPS estimator constructed on $\mathcal{D}$ which will have variance $n^2(\frac{1}{n} \cdot (1 - \frac{1}{n})) = n(1 - \frac{1}{n}) > 3$. However, returning back to our example, if we introduced a large (positive) bias to each segment in a data cube then queries like Q_1 which span multiple segments would accumulate bias from all of the segment summaries, limiting the final accuracy. Thus CoopStore intelligently allocates biases between the segments, balancing local reductions in variance with potential bias accumulation.

In general, if we have a segment $\mathcal{D}$ consisting of item weights $\{x_i \mapsto \delta_i\}$ then we bias the frequency estimates $\hat{f}_{\mathcal{D}}(x)$ by subtracting b from the count of every distinct element in $\mathcal{D}$ before constructing a PPS summary, and then adding b back to the stored weights. During PPS construction, h and thus the variance is reduced because $\mathcal{D}$ has a lower effective total weight $n_i[b]$ given by

$$n_i\,[b] = \sum_{x_i \in \mathcal{D}} (\delta_i - b)^+ \quad (16)$$

where $(x)^+$ is the positive part function $(x)^+ = \max(x, 0)$.

The error for a single segment $\mathcal{D}_i$ is now bounded by $\varepsilon_i \leq b_i + \nu_i$ where b_i is the bias and ν_i is the remaining unbiased PPS error on the bias-adjusted weights, so the MSRE for a query $\mathcal{Q}$ is:

$$E\left[\epsilon_{\mathcal{Q}}^2\right] \leq |\mathcal{Q}|^{-2} \left(\left(\sum_{\mathcal{D}_i \in \mathcal{Q}} b_i \right)^2 + \sum_{\mathcal{D}_i \in \mathcal{Q}} \frac{1}{4} \left(\frac{n_i\,[b_i]^2}{s_i^2} \right) \right) \quad (17)$$

Equation 16 shows that $n[b]$ is convex with respect to b since it is a sum of convex functions (max is convex), so the RHS of Equation 17 is convex as well.

Recap. In summary CoopStore does the following for cube aggregations.

1. Set summary sizes $s_i \propto \alpha_i^{1/3}$ using Equation 15, scaled so $\sum s_i = S_T$.
2. Solve for biases $\vec{b}$ that minimize the RHS of Equation 17 for $\mathcal{Q}$ a query over the entire dataset.
3. Construct PPS summaries according to $\vec{s}$ and $\vec{b}$

We optimize Equation 17 using the LBFGS-B solver [11]. To simplify computation we optimize b_i for a single aggregation:

the whole cube. An optimal setting of $\vec{b}$ for this whole cube query will not increase error over any other query compared to $\vec{b} = 0$.

5. EVALUATION

In our evalution, we show that:

1. Cooperative interval summaries achieve lower error as interval length increases compared with other summarization techniques: up to 8× for frequencies and 25× for quantiles (Section 5.2.1).

2. Cooperative cube summaries provide lower average error compared with alternative techniques, with reductions between 20% to 4.5× (Section 5.2.2).

3. Cooperative summary accuracy generalizes across different system and summary parameters, including accumulator size, maximum interval length, and workload specification (Sections 5.3.1 and 5.3.2).

4. Cooperative summaries introduce moderate query time (up to 3×) and significant construction time (over 90 seconds) overheads compared to existing mergeable summaries. (Section 5.4)

5.1 Experimental Setup

Error Measurement. Recall from Section 2.4 that we are interested in error bounds that are independent of a specific item or value x, so we evaluate the maximum error over x for a query $\mathcal{Q}$: $\epsilon_{\mathcal{Q}} := \max_x |\epsilon_{\mathcal{Q}}(x)|$. For a large domain of values U it is infeasible to compute $\max_{x \in U}$ so for frequency queries we estimate the maximum over a sample of 200 items drawn without replacement for item frequency queries and over 200 equally spaced quantiles from the complete dataset for quantile queries. Following common practice for approximate summaries [6], we scale the absolute count or rank error ε by the total size of the queried data to report normalized errors $\epsilon_{\mathcal{Q}} \cdot |\mathcal{Q}| = \varepsilon_{\mathcal{Q}}$.

Implementation. We evaluate implementations of Cooperative summaries (`Coop`) as part of our prototype system CoopStore written in Java with code available[1]. Runtime numbers when applicable are measured on an Intel Xeon 2.2Ghz machine[2]. Our prototype is a single node in-memory system though it can be extended to a distributed system in the same manner as Druid.

Our implementation of `CoopFreq` (Algorithm 1) uses $r = 1$ and sets h using `CalcT` from Section 4.1 rather than letting $h = |\mathcal{D}_t|/s$ which provides better segment accuracy while preserving the error bounds under a modified proof. We implement `CoopQuant` (Algorithm 2) with a cost function parameter α set based on a maximum interval length of $k_T = 1024$, and loss L calculated over the universe of elements seen so far when the full universe is not known ahead of time.

Datasets. We evaluate frequency estimates on 10 million destination ip addresses (`CAIDA`) from a Chicago Equinix backbone on 2016-01-21 available from CAIDA [12], 1 billion items (`Zipf`) drawn from a Zipf (Pareto) distribution with parameter $s = 1.1$, 32 million item purchases from the Intacart open dataset (`Instacart`) [1], and 10 million records from a production service request log at Microsoft with categorical item values for network service provider (`Provider`) and OS Build (`OSBuild`). We evaluate quantile estimates on 2 million active power readings (`Power`) from the UCI Individual household electric power consumption dataset [21], 10 million random values (`Uniform`) drawn from a continuous uniform $U \sim [0, 1]$ distribution, and 10 million records from the same Microsoft request log with numeric traffic values (`Traffic`). All of the the above have an associated sequential column for interval queries except for `Instacart`.

Table 3: Cube Datasets

Data	Dims	Segments	Summary Space
`Instacart`	4	10080	300000
`Zipf-C`, `Uniform-C`	4	10000	50000
`Traffic`	4	5938	50000
`OSBuild`, `Provider`	4	5938	100000

We evaluate cube queries on our datasets with categorical dimension columns, with parameters summarized in Table 3 and total space limit set to provide roughly consistent query error across the datasets. `Zipf-C` and `Uniform-C` consist of 10 million items from the `Zipf` and `Uniform` datasets associated with four dimension columns of 10 possible values each drawn from a zipf distribution with parameter $z = 1.0$.

5.2 Overall Query Accuracy

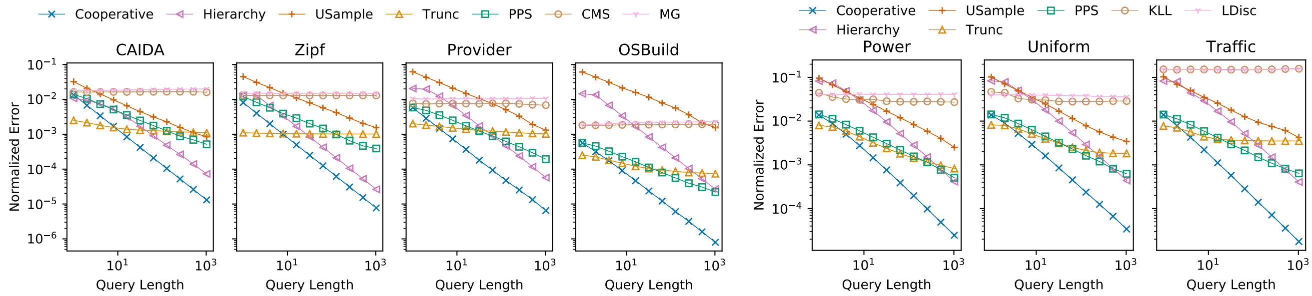
Summarization Methods. We compare a number of summarization techniques for frequencies and quantiles, and configure them to match total space usage.

We compare against three popular mergeable summaries: the optimal streaming quantiles sketch (`KLL`) from [31] and the low-discrepancy quantile sketch (`LDisc`) from [6] (the default quantiles sketch in Druid) both implemented by the Apache data sketches package [2], as well as the Count-Min frequency sketch (`CMS`) [19]. We also compare with the popular streaming (but not strictly mergeable) Misra-Gries sketch (`MG`) from [36] as implemented in the Apache data sketches package [2]. For the count-min sketch we set $d = 5$ and let the width $w = s$ parameter represent the space usage. We also compare against uniform random sampling (`USample`) [16], probability proportional to size sampling (`PPS`) [15], and optimal single-segment summaries (`Trunc`) that summarize a segment by storing the exact item counts for the top s items, or storing s equally spaced values for quantiles.

For interval queries we further compare with storing `Trunc` summaries in a hierarchy (`Hierarchy`) following [9, 18]. The `Hierarchy` summarization strategy with base b constructs h layers of summaries. Summaries in layer i are allocated space $b^i \cdot s_0$ to summarize aligned intervals of b^i segments. Any query interval of length k can be represented using $b\lceil \log_b k \rceil$ summaries from different layers. Since this requires maintaining $h = \log_b k_T$ layers, to fairly compare total space usage we scale the space s_0 allocated to the lowest layer summaries by a factor $s_0 = s / \log_b k_T$. Unless otherwise stated we use $b = 2$, though we will show in Section 5.3 that the choice does not have a significant impact on accuracy.

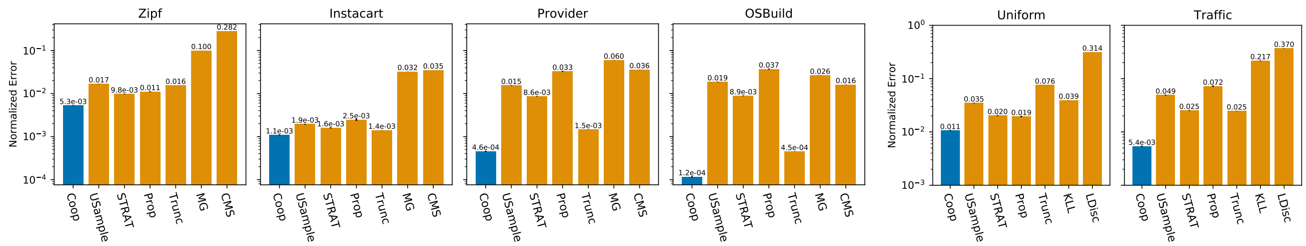
For cube queries we also compare with cube AQP techniques that use uniform `USample` with different space allocations: the `Prop` method uses `USample` summaries but allocates space proportional to each segment size as a global

[1] `https://github.com/stanford-futuredata/sketchstore`
[2] n1-highmem-32 on Google Cloud Compute



(a) Frequency Queries **(b)** Quantile Queries

Figure 5: Query error over interval queries of different lengths. Cooperative summaries have increasingly high accuracy as the query length increases.



(a) Frequency Queries **(b)** Quantile Queries

Figure 6: Average query error over a workload of cube queries. Cooperative summaries consistently provide lower average error than other AggPre summarization methods.

uniform random sample would, while the `STRAT` method uses the method in the STRAT AQP system [13], which like Cooperative summaries allocates space to minimize average error.

Query Processing. For all counter and sample-based summaries including `Coop`, `PPS`, `Trunc`, `USample`, and `Hierarchy`, we set the number of counters or samples to the same s and aggregate results from the summaries into an exact accumulator (a map from items to their cumulative counts). When accumulator memory is limited, we use a streaming sketch to accumulate results which introduces vanishing additional error as the size of the accumulator grows (Figure 9). For summaries with native merge routines including `KLL`, `LDisc`, `CMS`, and `MG`, we aggregate results by merging the summaries using their associated error-preserving merge routines [6].

5.2.1 Interval Queries

We first evaluate CoopStore accuracy on interval queries, partitioning datasets with associated time or sequence columns into $k_T = 2048$ size time segments. Then, we construct summaries with storage size $s = 64$.

In Figures 5a and 5b we show how relative query error ϵ_Q varies with the number of segments k spanned by the interval. For $k = 1, 2, 4, \ldots, 1024$ we sample 400 random start and end times for intervals with length k and plot the average and standard error of the query error.

Cooperative summaries outperform mergeable and other existing summaries as k increases. As the interval length increases, merging the mergeable summaries (`CMS`, `KLL`) and `MG` maintain their error as expected. Accumulating `Trunc` summaries also maintains the same constant error. `Hierarchy`, PPS, and `USample` are all able to reduce error when combining multiple summaries, while Cooperative summaries outperform all alternatives as k exceeds 10 summaries. We observe that despite our weaker worst-case bounds for cooperative quantile summaries, they achieve higher accuracy in practice compared to alternative methods. However, Cooperative summaries are not as accurate when aggregating less than 10 summaries compared to alternatives.

5.2.2 Cube Queries

For each of our data cube datasets we evaluate on a default query workload where each dimension has an independent $p = .2$ probability of being included as a filter, and if selected the dimension value is chosen uniformly at random. In Figures 6a and 6b we show the average relative error for frequency and quantile queries over 10000 random cube queries drawn from the specified workloads. We see that, on average, Cooperative summaries outperform alternative summarization techniques that allocate equal space to each segment, as well as uniform sampling techniques that optimize sample size allocation but do not use more sophisticated summaries or perform bias optimization.

In Figure 7 we break down the error for cube queries that filter on different numbers of dimensions on the `Uniform-C` and `Zipf-C` cube workloads. Cooperative summaries reduce the error for queries that filter on zero or one dimension. As a tradeoff Cooperative incurs higher error than other methods for queries with three or more filters. For a many workloads this tradeoff is desirable, and is configurable based on the user specified workload.

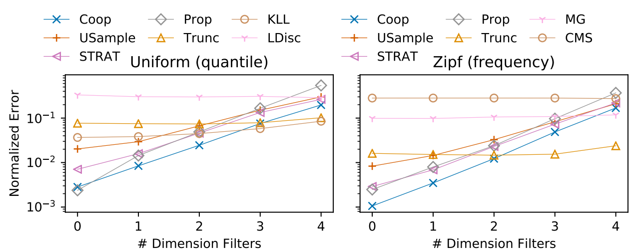


Figure 7: Query error broken down by number of dimension filters in a query. Cooperative summaries achieve lower error on queries that have fewer filters and aggregate more segments.

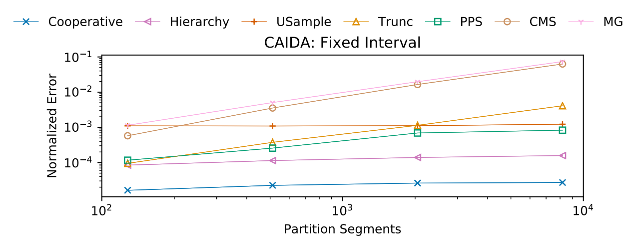


Figure 8: Varying the number of segments in a partitioning for a fixed interval query. As we increase the granularity of the partitioning, the query error for a fixed interval grows for most summaries, though Cooperative summaries remain more accurate than others.

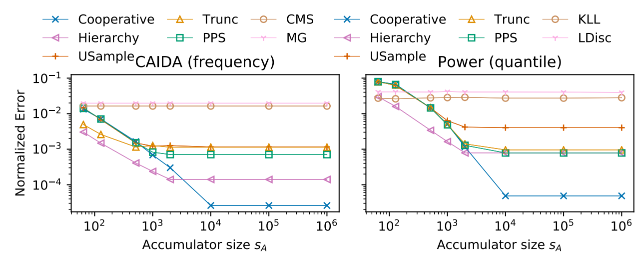


Figure 9: Query error as we vary the size of the accumulator s_A. For the large accumulators used in practice there is negligible additional error from the accumulator.

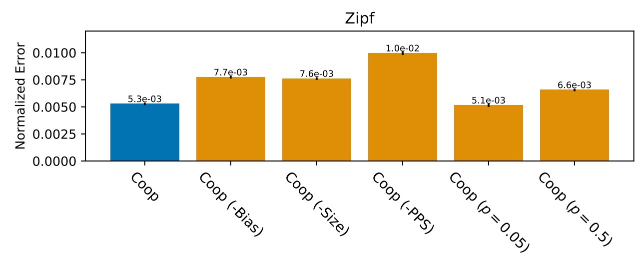


Figure 10: Lesion Study on `Zipf` cube optimizations. Removing any component reduces accuracy, though adjusting the workload parameter slightly improves accuracy.

5.3 Varying Parameters

Now we vary different system and summarization parameters to see their impact on accuracy, confirming that Cooperative summaries are able to provide improved accuracy under a variety of conditions.

5.3.1 *System Design*

The CoopStore system depends on a number of parameters. In this section we will show how accuracy varies with the granularity of partitioning datasets into segments, the accumulator size s_A, and the use of the size and bias optimizations Cooperative summaries use for data cubes.

Segment Granularity. CoopStore and other AggPre partition data into segments: more segments allows for more precise query conditions but constrains the size of each segment given total memory limits. In Figure 8 we measure the query error of interval frequency queries spanning a quarter of the `CAIDA` dataset when CoopStore partitions the data into varying numbers of segments. As the number of segments increases, query error for the same interval increases as well, though less so for Cooperative, `Hierarchy`, and uniform sampling than other summaries.

Finite Accumulator. In our evaluations for counter and sample-based summaries without a native merge routine, we accumulate results into an exact accumulator A that tracks items and weights. In settings where query memory is limited A would introduce an additional approximation error $\epsilon^{(A)} = 1/s_A$ which is negligible as $s_A \to \infty$.

In Figure 9 we illustrate how using accumulators of different sizes, affects final query accuracy on the `Power` and `CAIDA` datasets. For each size, we measure the error after accumulating 100 random interval aggregations spanning $k = 512$ segments. For the accumulators here we use SpaceSaving [35] for frequency queries and a streaming implementation of PPS (`VarOpt` [15]) for quantiles. $\epsilon^{(A)}$ goes to 0 as $s_A \to \infty$, and with at least 10 megabytes of memory available for s_A the additional error is negligible.

Cube Optimizer Lesion Study. In Figure 10 we show how the optimizations Cooperative summmaries (`Coop`) use for summarizing data cubes all play a role in providing high query accuracy by removing individual optimizations on the `Zipf` dataset. We experiment with removing the size optimizations (`Coop` (-Size)) and bias optimizations (`Coop` (-Bias)), and try replacing PPS summaries with uniform random samples (`Coop` (-PPS)). When, size optimization or bias optimization are removed, error increases, and similarly error increases when PPS summaries are replaced with uniform random samples.

Cube Workload Specification. We also evaluate how CoopStore accuracy depends on precise workload specification by constructing CoopStore instances configured for incorrectly specified workloads. Rather than the true $p = 0.2$ probability of including a dimension in the cube filter, we evaluate Cooperative summaries optimized for inaccurate workloads with $p = 0.05$ and $p = 0.50$. As seen in Figure 10, in both cases error remains below existing cube construction methods.

Interval Length Specification. For interval aggregations users specify a maximum expected interval length k_T. In Figure 11 we show the relative error for 20 random queries of length $k = 64$ as we vary k_T. All values $k_T \geq 64$ achieve good error and setting k_T much larger does not negatively

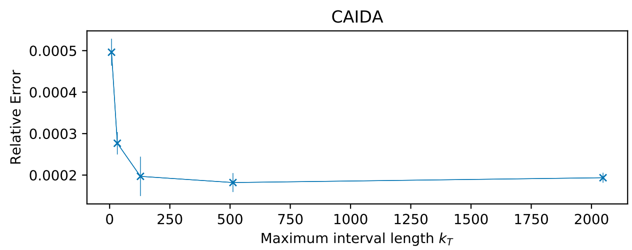


Figure 11: Error as we vary the maximum interval length parameter k_T. Overestimating k_T does not significantly change the quality of results.

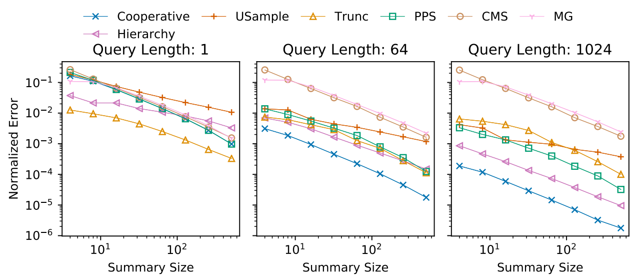


Figure 12: Query error as summary size changes. Cooperative summaries, like state of the art, have error $\epsilon = O(1/s)$

affect results. In practice accuracy is also robust to different values of k_T as long as it is conservatively longer than the expected queries.

5.3.2 Summary Design

Now we will examine how Cooperative summaries perform as individual segment summaries. The experiments below are run on the `CAIDA` dataset for interval aggregations.

Space Scaling. In Figure 12 we vary the space available to summaries for different interval lengths, confirming that like other state of the art summaries and sketches Cooperative and PPS summaries provide local segment error that scales inversely proportional to the space given, and maintain their accuracy under a wide range of summary sizes.

Hierarchical base b. Although `Hierarchy` summaries are parameterized by a base b, in Figure 13 we show that different values for b do not noticeably improve performance. Although there are improvements in optimizing b when merging small numbers ($k < 10$) of summaries, the difference is less than 10% for larger aggregations.

5.4 Runtime

Though Cooperative summaries and our AggPre prototype CoopStore are optimized to improve accuracy under memory constraints rather than to minimize runtime, in this section we evaluate their query time and construction time performance.

Query Time. In Figure 14 we evaluate the query time for interval queries on a representative subset of the datasets: omitted datasets show similar trends. Cooperative summaries and other summaries that make use of a precise accumulator have worse query time scaling than mergeable

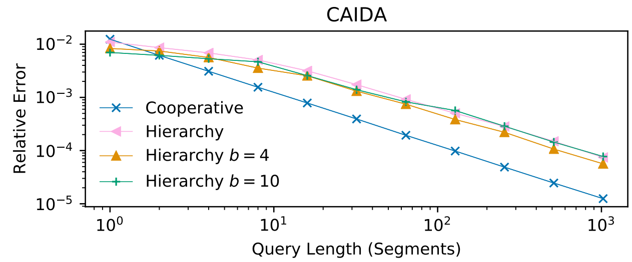
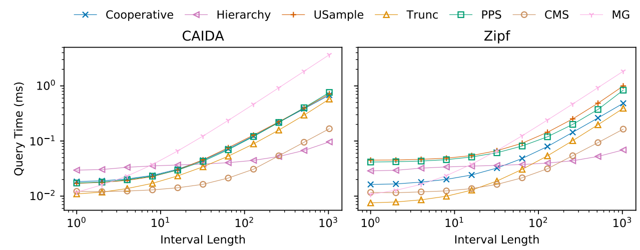
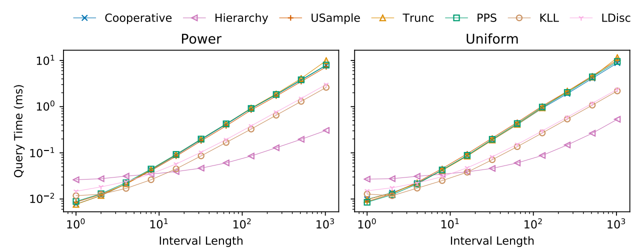


Figure 13: `Hierarchy` summary accuracy for different bases b. b does not have a large impact when accumulating across multiple summaries.



(a) Frequency Queries



(b) Quantile Queries

Figure 14: Query time over intervals. Cooperative summaries introduce an up to 3× overhead but runtimes remain below 10ms.

summaries like `CMS` as the accumulator grows over longer aggregations. Since Cooperative can provide dramatically improved accuracy and query time remains below 10ms, Cooperative summaries remain practical for systems like Druid. Query times over data cubes are similar.

Construction Time. To evaluate the construction time overheads of pre-computing different summaries, we measured the time taken to construct summaries over different datasets, excluding the time required to read raw records from disk. Tables 4 and 5 illustrate the time required to construct summaries over interval and data cube segments.

These overheads can be large but since we target settings where data loading is done using distributed batch processing systems they are not a significant bottleneck. Cooperative interval summaries require additional processing for tracking accumulated error that can result an roughly 2× construction overhead for frequencies and an up to 2000× overhead for quantiles compared to the fastest summaries. The overhead for constructing quantile summaries is high for the `Uniform` dataset since tracking the cumulative errors requires sorting very large sets of distinct values. Cooperative cube summaries require optimizing summary bias and

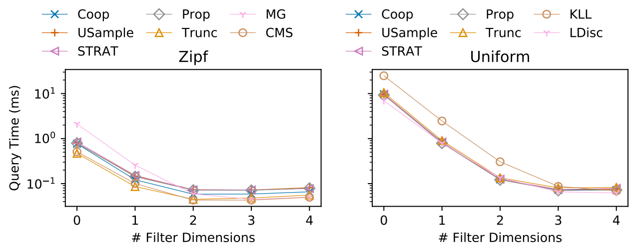


Figure 15: Query time over cubes. Cooperative summaries introduce an up to 2× runtime overhead.

Table 4: Interval Summaries Construction Time (ms)

sketch	Coop	Hierarchy	USample	Trunc	PPS	CMS	MG	KLL	LDisc
CAIDA	827	2741	521	564	479	588	644	–	–
Zipf	96K	505K	44K	51K	41K	64K	47K	–	–
Power	614	990	120	93	98	–	–	88	98
Uniform	89K	863	76	57	57	–	–	45	46

size allocation which results in an up to 3× construction overhead compared to the fastest summaries.

6. RELATED WORK

Precomputing Summaries. A number of existing approximate query processing (AQP) systems make use of precomputed approximate data summaries. An overview of these "offline" AQP systems can be found in [32], and they are an instance of the AggPre systems described in [39]. Like data cube systems they materialize partial results [27], but can support more complex query functions not captured by simple totals. Another class of systems use "online" AQP [28, 10, 43] and provide different latency and accuracy guarantees by computing approximations at query-time.

We are particularly motivated by Druid [48, 44] and similar offline systems [30] which aggregate over query-specific summaries for disjoint segments of data. However, these systems use mergeable summaries as-is, and do not optimize for improving accuracy under aggregation or take advantage of additional memory at query time to accumulate results more precisely. The authors in [49] apply hierarchical strategies to maintain summary collections for interval queries but like mergeable summaries maintain do not reduce error when combining summaries. Systems like BlinkDB [7], STRAT [13], and AQUA [5] maintain random stratified samples to support general-purpose queries. Our choice of minimizing mean squared error over a workload follows the setup in STRAT [13]. However, individual simple random samples are not as accurate as specialized frequency or quantile summaries [37].

Techniques for summarizing hierarchical intervals [9] are complementary, but incur additional storage overhead making them less accurate than Cooperative summaries and scale poorly to cubes with multiple dimensions [42].

Streaming and Mergeable Summaries. Many compact data summaries are developed in the streaming literature [26, 36, 19, 31], including summaries for sliding windows [8]. However, the standard streaming model generally assumes limited working memory during summary construction [38]. Mergeable summaries [6] allow combining multiple summaries but require that intermediate results take up

Table 5: Cube Summaries Construction Time (ms)

sketch	Coop	USample	STRAT	Prop	Trunc	MG	CMS	KLL	LDisc
Zipf-C	1177	647	681	650	659	378	639	–	–
Provider	489	115	133	114	107	249	593	–	–
Uniform-C	765	720	763	726	715	–	–	500	542
Traffic	417	390	411	389	386	–	–	462	478

no more space than the inputs, and thus merely maintain relative error under merging. Other work targeting AggPre systems has focused on improving summary update and merge runtime performance [24, 34] rather than improving the accuracy of query results.

Other Summarization Models. The CoopStore model, where more memory is available for construction and aggregation than for storage, is closer to the model used in non-streaming settings including discrepancy theory and communication theory.

Coresets and ϵ-approximations are data structures for approximate queries that allow more resource-intensive precomputation and aggregation [41]. ϵ-approximations are part of discrepancy theory which attempts to approximate an underlying distribution with proxy samples [14]. We draw inspiration from discrepancy theory to manage error accumulation in our cooperative summaries, especially the results in [46] which pioneered the use of the cosh cost function. Other work in this area minimize error accumulation along multiple dimensions [40]. However, we are not away of coreset or ϵ-approximations that allow for complex queries CoopStore supports: quantiles and item frequencies over multiple data segments. and cube aggregations. In particular, existing work supporting range queries [40] do not provide per-segment local guarantees. Recent work developing hierarchical histograms [45] optimize size allocation among histograms similar to our Cooperative cube summaries and [13], but target range queries and do not address per-segment error for quantiles and item frequencies.

Work in communication theory and distributed streaming assume the network is a bottleneck when aggregating results. There is existing work analyzing how multiple random samples can be combined to reduce aggregate error in this setting [50, 51]. However, in communication theory the samples are constructed per-query, while CoopStore precomputes summaries that can be used for arbitrary future queries. Furthermore random samples are not as space efficient as Cooperative summaries.

Related techniques in differential privacy [42, 18] and matrix rounding [20] consider approximate representations of data segments for the purposes of privacy, but do not explicitly optimize for space or support heavy hitters and quantile queries.

7. CONCLUSION

CoopStore uses Cooperative summaries that are optimized to reduce query error over large aggregations. Cooperative summaries take advantage of additional memory resources available at summary construction and aggregation and target a common class of structured frequency and quantile queries. These summaries can thus efficiently serve a range of monitoring and data exploration workloads.

Acknowledgments

This research was made possible with feedback and assistance from our collaborators at Microsoft including Atul Shenoy, Asvin Ananthanarayan, and John Sheu, as well as collaborators at Imply including Gian Merino. This research was supported in part by affiliate members and other supporters of the Stanford DAWN project – Ant Financial, Facebook, Google, Infosys, NEC, and VMware – as well as Toyota Research Institute, Northrop Grumman, Cisco, SAP, and the NSF under CAREER grant CNS-1651570. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. Toyota Research Institute ("TRI") provided funds to assist the authors with their research but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity.

8. REFERENCES

[1] The instacart online grocery shopping dataset 2017, 2017. `https://www.instacart.com/datasets/grocery-shopping-2017`.

[2] Apache data sketches library, 2020. `https://datasketches.apache.org/`.

[3] Datasketches quantiles sketch module. `https://druid.apache.org/docs/latest/development/extensions-core/datasketches-quantiles.html`, 2020.

[4] Quantiles accuracy and size. `https://datasketches.apache.org/docs/Quantiles/QuantilesAccuracy.html`, 2020.

[5] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *SIGMOD*, pages 574–576, 1999.

[6] P. K. Agarwal, G. Cormode, Z. Huang, J. Phillips, Z. Wei, and K. Yi. Mergeable summaries. In *PODS*, 2012.

[7] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: Queries with bounded errors and bounded response times on very large data. In *EuroSys*, pages 29–42, 2013.

[8] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *PODS*, pages 286–296, 2004.

[9] R. B. Basat, R. Friedman, and R. Shahout. Stream frequency over interval queries. *PVLDB*, 12(4):433–445, 2018.

[10] M. Budiu, P. Gopalan, L. Suresh, U. Wieder, H. Kruiger, and M. K. Aguilera. Hillview: A trillion-cell spreadsheet for big data. *PVLDB*, 12(11):1442–1457, July 2019.

[11] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, 16(5):1190–1208, 1995.

[12] CAIDA. The caida ucsd anonymized internet traces, 2016. `http://www.caida.org/data/passive/passive_dataset.xml`.

[13] S. Chaudhuri, G. Das, and V. Narasayya. Optimized stratified sampling for approximate query processing. *ACM Trans. Database Syst.*, 32(2), 2007.

[14] B. Chazelle. *The Discrepancy Method: Randomness and Complexity*. Cambridge University Press, New York, NY, USA, 2000.

[15] E. Cohen, G. Cormode, and N. G. Duffield. Structure-aware sampling: Flexible and accurate summarization. *PVLDB*, 4(11):819–830, 2011.

[16] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1–3):1–294, 2012.

[17] G. Cormode and M. Hadjieleftheriou. Methods for finding frequent items in data streams. *The VLDB Journal*, 19(1):3–20, Feb 2010.

[18] G. Cormode, T. Kulkarni, and D. Srivastava. Answering range queries under local differential privacy. *PVLDB*, 12(10):1126–1138, 2019.

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

[20] B. Doerr, T. Friedrich, C. Klein, and R. Osbild. Unbiased matrix rounding. In L. Arge and R. Freivalds, editors, *Algorithm Theory – SWAT 2006*, pages 102–112, 2006.

[21] D. Dua and C. Graff. UCI machine learning repository, 2017.

[22] T. Dunning and O. Ertl. Computing extremely accurate quantiles using t-digests. *arXiv preprint arXiv:1902.04023*, 2019.

[23] E. Gan, P. Bailis, and M. Charikar. Coopstore: Optimizing precomputed summaries for aggregation. Technical report, 2020. `http://edgan8.github.io/assets/papers/coopstore-revise.pdf`.

[24] E. Gan, J. Ding, K. S. Tai, V. Sharan, and P. Bailis. Moment-based quantile sketches for efficient high cardinality aggregation queries. *PVLDB*, 11(11):1647–1660, July 2018.

[25] 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.

[26] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD*, volume 30, pages 58–66, 2001.

[27] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD*, pages 205–216, 1996.

[28] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *SIGMOD*, pages 171–182, 1997.

[29] C.-T. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in olap data cubes. *SIGMOD*, 26(2):73–88, 1997.

[30] J.-F. Im, K. Gopalakrishna, S. Subramaniam, M. Shrivastava, A. Tumbde, X. Jiang, J. Dai, S. Lee, N. Pawar, J. Li, and R. Aringunram. Pinot: Realtime olap for 530 million users. In *SIGMOD*, pages 583–594, 2018.

[31] Z. Karnin, K. Lang, and E. Liberty. Optimal quantile approximation in streams. In *FOCS*, pages 71–78, 2016.

[32] K. Li and G. Li. Approximate query processing: What is new and where to go? *Data Science and Engineering*, 3(4):379–397, Dec 2018.

[33] G. Luo, L. Wang, K. Yi, and G. Cormode. Quantiles over data streams: experimental comparisons, new analyses, and further improvements. *VLDB*, 25(4):449–472, 2016.

[34] C. Masson, J. E. Rim, and H. K. Lee. Ddsketch: A fast and fully-mergeable quantile sketch with relative-error guarantees. *PVLDB*, 12(12):2195–2205, Aug. 2019.

[35] A. Metwally, D. Agrawal, and A. El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *Proceedings of the 10th International Conference on Database Theory*, ICDT'05, pages 398–412, 2005.

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

[37] B. Mozafari and N. Niu. A handbook for building an approximate query engine. *IEEE Data Eng. Bull.*, 38(3):3–29, 2015.

[38] S. Muthukrishnan et al. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):117–236, 2005.

[39] J. Peng, D. Zhang, J. Wang, and J. Pei. Aqp++: Connecting approximate query processing with aggregate precomputation for interactive analytics. In *SIGMOD*, page 1477–1492, 2018.

[40] J. M. Phillips. Algorithms for ϵ-approximations of terrains. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming - Volume Part I*, ICALP '08, pages 447–458, 2008.

[41] J. M. Phillips. Coresets and sketches. In C. D. Toth, J. O'Rourke, and J. E. Goodman, editors, *Handbook of Discrete and Computational Geometry*, chapter 48, pages 1267–1286. CRC Press, 2017.

[42] W. Qardaji, W. Yang, and N. Li. Understanding hierarchical methods for differentially private histograms. *PVLDB*, 6(14):1954–1965, Sept. 2013.

[43] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman. Aggregation and degradation in jetstream: Streaming analytics in the wide area. In *NSDI*, pages 275–288, 2014.

[44] N. Ray. The art of approximating distributions: Histograms and quantiles at scale. `http://druid.io/blog/2013/09/12/the-art-of-approximating-distributions.html`, 2013.

[45] M. Shekelyan, A. Dignos, and J. Gamper. Digithist: A histogram-based data summary with tight error bounds. *PVLDB*, 10(11):1514–1525, Aug. 2017.

[46] J. Spencer. Balancing games. *Journal of Combinatorial Theory, Series B*, 23(1):68 – 74, 1977.

[47] D. Ting. Data sketches for disaggregated subset sum and frequent item estimation. In *SIGMOD*, pages 1129–1140, 2018.

[48] F. Yang, E. Tschetter, X. Léauté, N. Ray, G. Merlino, and D. Ganguli. Druid: A real-time analytical data store. In *SIGMOD*, pages 157–168, 2014.

[49] K. Yi, L. Wang, and Z. Wei. Indexing for summary queries: Theory and practice. *ACM Trans. Database Syst.*, 39(1), Jan. 2014.

[50] Zengfeng Huang, Ke Yi, Yunhao Liu, and Guihai Chen. Optimal sampling algorithms for frequency estimation in distributed data. In *2011 Proceedings IEEE INFOCOM*, pages 1997–2005, 2011.

[51] Q. G. Zhao, M. Ogihara, H. Wang, and J. J. Xu. Finding global icebergs over distributed data sets. In *PODS*, page 298–307, 2006.

APPENDIX

A. PPS SUMMARIES

PPS summaries for item frequencies and ranks have been studied in the sampling literature, and in this section we reproduce implementation details relevant to our use of PPS summaries in CoopStore.

In Algorithm 3 we present a known procedure to set h as low as possible to minimize error while keeping the summary size of a PPS summary at most s (Algorithm 4, Stream-τ in [15])

Algorithm 3 Calculate minimal h threshold

```
function CalcT(D, s)
    h ← |D|/s
    H ← {}                                   ▷ Local Heavy Hitters
    while max_{x∈D\H} f_D(x) ≥ h do
        x_max ← arg max_{x∈D\H} D(x)
        H ← H ∪ {x_max}
        h ← (Σ_{x∈D\H} f_D(x)) / (s − |H|)
    return h
```

One way to implement PPS is to independently sample items according to Equation 12, but this does not guarantee the summary will store exactly s values. Instead we use the `PairAgg` procedure in Algorithm 4 to transform sampling probabilities for pairs of items until we have s or $s-1$ values with probability 1. We can do so in a way that guarantees that the error $\max_x |\varepsilon(x)| \leq h$ and is unbiased with $E[\varepsilon(x)] = 0$ for both frequency and rank queries. See [15] for details.

Algorithm 4 Pair Aggregation for PPS

```
function PairAgg(p_i, p_j)
    if p_i + p_j < 1 then
        if rand() < p_i/(p_i + p_j) then p_i ← p_i + p_j; p_j ← 0
        else p_j ← p_i + p_j; p_i ← 0
    else
        if rand() < (1 − p_j)/(2 − p_i − p_j) then p_i ← 1; p_j ← p_i + p_j − 1
        else p_i ← p_i + p_j − 1; p_j ← 1
```

B. COOPERATIVE SUMMARY PROOFS

Lemma 1.

PROOF. Recall that we have a segment

$$\mathcal{D}_t = \{x_1 \mapsto \delta_1, \ldots, x_r \mapsto \delta_r\}.$$

Let H be the set of local heavy hitters $H = \{x_i : \delta_i \geq h\}$ and let $U' = U \setminus H$ be the remaining items. We can decompose our summary as $S_t = S_H \cup S_V$ where $V = S_t \setminus H$.

$$S_H = \{x_i \mapsto \delta_i : x_i \in H\} \quad (18)$$
$$S_V = \{x_i \mapsto \min(\varepsilon_{t-1}(x_i) + \delta_i, rh) : x_i \in V\}. \quad (19)$$

This keeps $\varepsilon_t(x) \geq 0$ across segments, i.e. our estimates are always underestimates.

Let $G = L_t - L_{t-1} = \sum_{x_i \in U} [\phi(\varepsilon_t(x_i)) - \phi(\varepsilon_{t-1}(x_i))]$ where $\phi(z) = \exp(\alpha z)$. For heavy hitters $\varepsilon_t(x_i) = \varepsilon_{t-1}(x_i)$ so they do not change the cumulative cost L_t.

$$G = \sum_{x_i \in V} [\phi(\max(\varepsilon_{t-1}(x_i) + \delta_i - rh, 0)) - \phi(\varepsilon_{t-1}(x_i))] + \sum_{x_i \in U' \setminus V} [\phi(\varepsilon_{t-1}(x_i) + \delta_i) - \phi(\varepsilon_{t-1}(x_i))]$$

Simplifying using $\max(0, y) = y + (0-y)1_{y \leq 0}$ and $\phi(x+y) = \phi(x)\phi(y)$:

$$\begin{aligned} G = &\sum_{x_i \in U' \setminus V} \phi(\varepsilon_{t-1}(x_i) + \delta_i) [1 - \phi(-\delta_i)] \\ &+ \sum_{x_i \in V} \phi(\varepsilon_{t-1}(x_i) + \delta_i) [\phi(-rh) - \phi(-\delta_i)] \\ &+ \sum_{x_i \in V} [\phi(0) - \phi(\varepsilon_{t-1}(x_i) + \delta_i - rh)] \cdot 1_{\varepsilon_{t-1} + \delta_i \leq rh} \end{aligned}$$

For non-heavy hitters, Algorithm 1 selects items in V with the highest $\varepsilon_{t-1}(x_i) + \delta_i$. If we let $\ell = \arg\min_{x_i \in V} \varepsilon_{t-1}(x_i) + \delta_i$ then

$$\begin{aligned} \forall x_i \in V \quad & \varepsilon_{t-1}(x_\ell) + \delta_\ell \leq \varepsilon_{t-1}(x_i) + \delta_i \\ \forall x_i \in U' \setminus V \quad & \varepsilon_{t-1}(x_\ell) + \delta_\ell \geq \varepsilon_{t-1}(x_i) + \delta_i. \end{aligned}$$

Technical but standard applications of the inequalities $\phi(x) \geq 1 + ax$, and $\phi(x) \leq 1 + \alpha x + \alpha^2 x^2/2$ for $x \leq 0$ yields:

$$G \leq \phi(\varepsilon_{t-1}(x_\ell) + \delta_\ell)|V| \left[\alpha h - \alpha hr + \alpha^2 h^2 r^2/2\right] + \alpha rh|V|$$

$|V| \leq s$ and $h \leq |\mathcal{D}_t|/s$ so when $\alpha \leq \frac{2}{h}\frac{r-1}{r^2}$, $G \leq \alpha r|\mathcal{D}_t|$ $\square$

Lemma 2.

Proof. First note that the choice of which element z_j is chosen from each chunk for inclusion in the summary sample S_t does not affect $\varepsilon_t(x)$ for x outside the chunk $\mathcal{D}_{t,j}$ so we can consider the choices independently. This is because the selected element is assigned a proxy count equal to the population of the whole chunk $h = |\mathcal{D}_{t,j}| = |\mathcal{D}_t|/s$.

Let $L_{t,j} := \sum_{x_i \in \mathcal{D}_{t,j}} \phi(\varepsilon_t(x_i))$ be total cost for chunk j. Since Algorithm 2 selects a value z that minimizes L_t, the final value for $L_{t,j}$ must be lower than any weighted average of the possible $L_{t,j}$ for different choices of x.

$$L_{t,j} \leq \sum_{z \in \mathcal{D}_{t,j}} \frac{f_{\mathcal{D}_t}(z)}{h} \left[\sum_{x \in \mathcal{D}_{t,j}} \phi\left(\varepsilon_{t-1}(x) + r_{\mathcal{D}_{t,j}}(x) - 1_{x \geq z} h\right)\right]$$

Abbreviate $p_x := \frac{1}{h} r_{\mathcal{D}_{t,j}}(x) = \frac{1}{h}\sum_{x_i \in \mathcal{D}_{t,j}} \delta_i \cdot 1_{x_i \leq x}$. Switching the order of summation gives:

$$\begin{aligned} L_{t,j} \leq \sum_{x \in \mathcal{D}_{t,j}} &[p_x \phi(\varepsilon_{t-1}(x) + hp_x - h) \\ &+ (1 - p_x)\phi(\varepsilon_{t-1}(x) + hp_x)] \end{aligned}$$

Now we can make use of Lemma 3 below to simplify

$$L_{t,j} \leq \exp(\alpha^2 h^2/2) L_{t-1,j}$$

Finally, since $L_t = \sum_{j=1}^{s} L_{t,j}$ we have the lemma. $\square$

Lemma 3 can be proven using the cosh angle addition formula and Taylor expansions.

Lemma 3. *For $0 \leq p \leq 1$ and $t \geq 0$*

$$p\cosh(x + t(p-1)) + (1-p)\cosh(x + tp) \leq \exp(t^2/2)\cosh(x) \quad (20)$$

Proof. We abbreciate cosh, sinh as c, s and the left hand side of Equation 20 as LHS. Using the angle addition formula:

$$\begin{aligned} LHS = &\, p\,[c(x)c(t(p-1)) + s(x)s(t(p-1))] \\ &+ (1-p)\,[c(x)c(tp) + s(x)s(tp)] \end{aligned}$$

Then since $s(x) \leq c(x)$:

$$\begin{aligned} LHS &\leq pc(x)\,[c(t(p-1)) + s(t(p-1))] \\ &\quad + (1-p)c(x)\,[c(tp) + s(tp)] \\ &= c(s)\,[p\exp(t(p-1)) + (1-p)\exp(tp)] \end{aligned}$$

We now consider two cases: $t < 2$ and $t \geq 2$.

If $t < 2$, we expand out taylor series to get that:

$$\begin{aligned} p\exp(t(p-1)) + (1-p)\exp(tp) &\leq 1 + \frac{t^2}{2}(p(1-p)) \cdot 3 \\ &\leq 1 + t^2/2 \leq \exp(t^2/2) \end{aligned}$$

If $t \geq 2$ then

$$\begin{aligned} p\exp(t(p-1)) + (1-p)\exp(tp) &\leq \exp(tp) \\ &\leq \exp(t) \leq \exp(t^2/2) \end{aligned}$$

In either case we can conclude that:

$$LHS \leq \cosh(x)\exp(t^2/2)$$

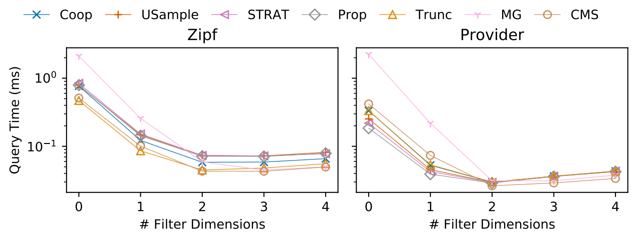
$\square$

B.1 Error Lower Bounds

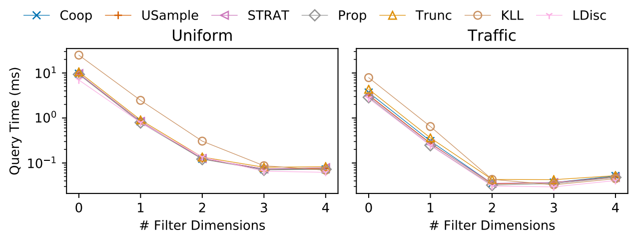
In this section we will provide details on an adversarial dataset for which no online selection of items for a counter-based summary can achieve better than absolute $\varepsilon = \Omega(\log k)$ error for item frequency queries.

Theorem 3. *There exists a sequence of $k = 2^{h+1}$ data segments $\mathcal{D}_i$ consisting of $|\mathcal{D}_i| = 2s$ item values each such that for all possible selections of s items for counter-based summaries S_i, $\exists x.|f_{\mathcal{D}_i,\ldots,\mathcal{D}_k}(x) - \hat{f}_{S_i,\ldots,S_k}(x)| \geq h$.*

Proof. Consider a universe of item values $U = 1, \ldots, 2s2^h$. For $i = 1, \ldots, 2^h$ let $\mathcal{D}_i = \{2s(i-1)+1, \ldots, 2si\}$ where each item occurs at most once. Since each summary S_i can only store s item values, there must be a set of $s2^h$ items (U_1) that are not stored in any summary, but that have appeared at least once in the data. Now let the next 2^{h-1} data segments $\mathcal{D}_i$ for $i \geq 2^h + 1$ constain $2s$ distinct item values each from U_1. Again, since each summary can only store s item values now there must be a set of $s2^{h-1}$ items (U_2) that are not stored in any summary, but that have appeared twice in the data. This repeats for for increasing U_i: at each stage U_i we have data segments come in that contain only items



(a) Frequency Queries



(b) Quantile Queries

Figure 16: Query time over cubes. Cooperative summaries introduce an up to 2× overhead over the fastest mergeable summaries.

the summaries have not been able to store, until we have at least one item not stored in any summary but that has appeared $h+1$ times in the data. $\square$

C. ADDITIONAL EVALUATIONS

Query Time over Cubes. In Figure 16 we show the query times for different summary types on additional data cube datasets.