Storyboard: Optimizing Precomputed Summaries for Aggregation*

Edward Gan, Peter Bailis, Moses Charikar
Stanford University

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 Storyboard, a query system that optimizes item frequency and quantile summaries for accuracy when aggregating over multiple segments. Compared to conventional mergeable summaries, Storyboard 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 4.4× over data cube aggregations on industrial datasets compared to standard summarization methods, with provable worst-case error guarantees.

1 INTRODUCTION

An emerging class of data systems precompute aggregate summaries over a dataset to reduce query times. These precomputation (AggPre [38]) 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 [28, 47]. They can then process queries by aggregating results from the segment summaries. Unlike traditional data cube systems [23], the summaries go beyond scalar counts and sums and include data structures that can approximate quantiles and frequent items [15]. 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 [21, 22] 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 [43].

Although querying summaries is more efficient than querying raw data, precomputing summaries also limits query accuracy. Given a total storage budget and many data segments, each segment summary in an AggPre system has limited storage space – often <10 kilobytes – and thus limited accuracy [1]. Prior work on *mergeable summaries* introduces summaries that can be combined with no loss in accuracy, and are commonly used in AggPre systems [5, 22, 43]. However, even mergeable summaries have maximum accuracy capped 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.

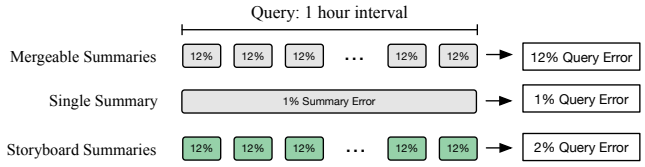


Figure 1: Given a space budget, mergeable summaries preserve accuracy when combined but cannot match the accuracy of using a single larger summary. Storyboard closes the gap by optimizing summaries for accurate aggregations.

In this paper we introduce Storyboard, an AggPre query system that optimizes frequent items and quantile summaries for aggregation. Unlike mergeable summaries, Storyboard queries that combine results from multiple summaries have lower relative error than any summary individually. To do so Storyboard uses a different resource model than most existing summaries were designed for. While mergeable 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. Storyboard takes advantage of these additional resources to construct summaries that

*Preprint. Under review.

compensate for the errors in other summaries they may be combined with, and then aggregate results using a large and precise accumulator.

Storyboard focuses on supporting 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` [8], while data cube queries aggregate over data matching specific dimension values, such as `loc=USA AND type=TCP` [23]. When aggregating k summaries together, Storyboard can reduce relative error by nearly a factor of k for interval aggregations and a factor of $\sqrt{k}$ for other aggregations, compared with no reduction in error for mergeable summaries. These two query types cover a wide class of common queries and Storyboard can construct summaries optimized for either of the two types.

Interval Queries. For interval queries, Storyboard uses novel summarization techniques which we call *cooperative* summaries. Cooperative summaries 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 (heavy hitters) summaries have tended to underestimate 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 smaller than existing summarization techniques. Hierarchical approximation techniques [8, 41] 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 [49], and for frequencies exceed the accuracy of state of the art hierarchical approaches [8]. Empirically, cooperative summaries provide a 4-25× reduction in error on 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 Storyboard used randomized weighted samples and reduces error further by optimizing for an expected workload of queries. Storyboard exploits the fact that data cubes often have dimensions with skewed value distributions: some values or combination of values that occur far more frequently than others. Then, Storyboard optimizes the allocation of storage space and introduces targeted biases where they will have the greatest impact to minimize average query error. Empirically, these optimizations yield an up to 4.4× reduction in average error compared with standard data cube summarization techniques.

In summary, we make the following contributions:

(1) We introduce Storyboard, an approximate AggPre system that provides improved query accuracy for aggregations by taking advantage of additional memory resources at data ingest and query time.

(2) We develop cooperative frequency and quantile summaries that minimize error when aggregating over intervals and establish worst-case bounds on their error.

(3) We develop techniques for allocating space and bias among randomized summaries to minimize average error under cube aggregations.

The remainder of the paper proceeds as follows. In Section 2 we provide motivating context. In Section 3 we present Storyboard and its query model. In Section 4 we describe cooperative summaries optimized for intervals. In Section 5 we describe optimizations for data cubes. In Section 6 we evaluate Storyboard's accuracy. We describe related work in Section 7 and conclude in Section 8.

2 BACKGROUND

Storyboard targets aggregation queries common in monitoring and data exploration applications, and improves accuracy for these queries by taking advantage of memory resources that real world summary precomputation systems already have available at query and ingest time. This motivation comes out of our experience collaborating with engineers at Imply, the developers of Druid, as well as a cloud services team at Microsoft.

Aggregation Queries. Existing usage of AggPre systems feature queries that span many segment summaries but follow structured patterns. At a cloud services team at Microsoft, users interacted with a Druid-like system primarily through a time-series monitoring dashboard which they used to track trends and explore anomalies. In addition to count and average queries, Top-K (heavy hitters) item frequency and quantile queries were prevalent. The most common forms of aggregations were over time intervals and grouped cube roll-ups. For instance, engineers often wanted to see the most common ip address frequencies over specific release windows or server configurations.

Notably, the queries over time intervals and data cubes involved combining results from a large number of summaries. In Figure 2 we describe a set of 33K Top-K item frequency queries and 130K quantile queries issued to an AggPre system at Microsoft. More than half of the queries span intervals longer than a day. Since the system stored summaries at a 5 minute granularity, this meant combining results from hundreds of summaries or reverting to less accurate, coarser grained roll-ups. Data cube roll-ups were also common. Users would commonly issue queries that filtered or grouped on a

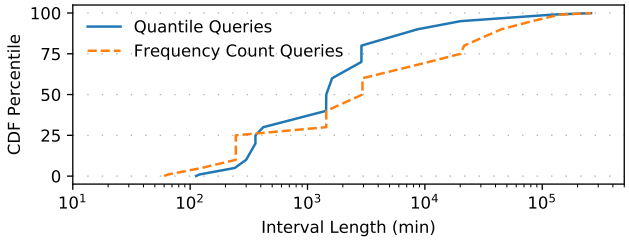


Figure 2: Distribution of user-issued time interval queries to a Druid-like system at Microsoft. More than half of the queries span > 100 five-minute segments.

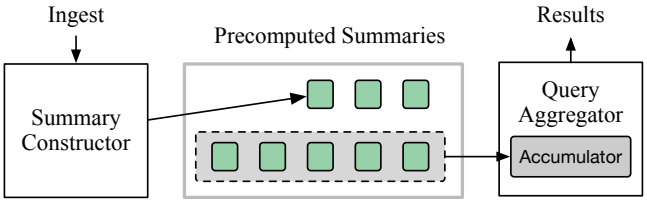


Figure 3: Storyboard precomputes summaries at ingest optimized to minimize error under aggregations. At query time, results from multiple summaries are combined using a precise accumulator to provide accurate results.

dimension, and "drill-down" [23] as needed, sometimes as part of anomaly explanation systems like MacroBase [2, 3]. Over 50% of all cubes had more than 10K dimension value segments and queries that span hundreds of cube segments were common.

Storyboard thus optimizes for the accuracy of time interval and cube queries that span not just a single summary, but aggregate over many summaries.

Memory Constraints. At both Imply and Microsoft, the storage space available to each summary was limited. Since summaries are maintained for each of potentially millions of segments, and nodes have limited memory and cache, the memory must be divided amongst the segment summaries. To illustrate, by default each quantile summary in Druid is configured for 2% error and roughly 10 kB of memory usage [1]. Thus it is important to use summaries that provide high accuracy with minimal storage overhead. However, in real-world deployments there is much more memory available during summary construction and aggregation than there is for storage. For instance, in Druid the use of Hadoop map-reduce jobs for batch data ingestion allows for effectively unconstrained memory limits during summary construction. Then, at query time, engineers at Imply report that a standard deployment uses query processing jobs with 0.5 GB of memory each when aggregating summaries together.

While standard mergeable summaries [5] are designed to maintain the same memory footprint during construction and aggregation that they use in storage, Storyboard takes advantage of additional memory at both construction and query time to achieve higher query accuracy.

3 SYSTEM DESIGN

In this section we describe Storyboard's system design. We discuss the types of queries supported, how summaries are constructed for different query types, and how summaries are aggregated to provide accurate query results. We outline the system components in Figure 3.

3.1 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 Storyboard query $g_Q(x)$ specifies an aggregation of records Q and a function g to estimate for the value x. Q defines an *aggregation* with a selection condition: either a one-dimensional interval or a multi-dimensional cube query [8, 23].

Definition 1. *An interval aggregation specifies* $Q^{(time)} = \{\rho : T_0 \leq t < T_1\}$ *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* $Q^{(cube)} = \{\rho : d_{i_1} = v_{i_1} \wedge \ldots \wedge d_{i_k} = v_{i_k}\}$ *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 [16, 32]. 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 denote either frequencies or ranks.

$$f_Q(x) = \sum_{\rho_i \in Q} 1_{x_i = x} \qquad r_Q(x) = \sum_{\rho_i \in Q} 1_{x_i \leq x}. \quad (1)$$

Using these primitives, Storyboard can also return estimates for quantiles and Top K / Heavy Hitters queries, which we will discuss in more detail in Section 3.3.

3.2 Data Ingest

Before Storyboard can ingest data, users specify whether they want the dataset to support interval or data cube aggregations, and whether they want the dataset to support 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.

Storyboard 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 $\mathcal{D}_{\vec{v}} = \{\rho : d_1 = v_1 \wedge \cdots \wedge d_{m_d} = v_{m_d}\}$. Once the dataset is partitioned we can represent the records in each segment as mappings from values to counts:

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

For each segment $\mathcal{D}$ Storyboard constructs a summary S consisting of s value, count mappings

$$S \coloneqq \{x_1 \mapsto \gamma_1, \ldots, x_s \mapsto \gamma_s\}.$$

This is similar to other counter based summaries [16, 34] and weighted sampling summaries [49]. Unlike tabular sketches such as the Count-Min Sketch [18] Storyboard summaries include the values x. We assume we have enough memory and compute to generate S, making our routines closer to coreset construction [40] than streaming sketches [37]. More details on how the values x and counts γ are chosen for each summary are given in Section 4.1 for interval aggregations and Section 5.1 for cube aggregations.

3.3 Query Processing

After the summaries have been constructed, the Storyboard query processor can return query estimates $\hat{g}_Q(x)$ for different aggregations 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 Q by adding up the estimates for the segment summaries.

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

$$\hat{g}_Q(x) = \sum_{S_i \in Q} g_{S_i}(x) \tag{2}$$

For single rank or frequency estimates $\hat{g}_Q(x)$ the query processor can precisely add up scalar estimates using Equation 2. This is more efficient than merging mergeable summaries [5], since we are just accumulating scalars, and potentially more accurate as we will discuss in Section 3.4.

Storyboard uses a richer accumulator data structure A to support quantile and top-k / heavy hitter queries. When there is sufficient memory, A tracks the proxy values and counts in $S_1, \ldots, S_k$. We then sort the items in A by value to estimate quantiles or by count to estimate heavy hitters. 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 [34] for heavy hitters and a PPS (`VarOpt` [14]) sample for quantiles. Then we can query A to get quantile or heavy hitters estimates. 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 deployment described in Section 2.

3.4 Error Model

Consider the absolute (i.e. unscaled) error ε_Q which is the difference between the true and estimated item frequency counts, or the difference between the true and estimated ranks for a query aggregation Q. Throughout the paper, we will use absolute errors ε for analysis, when comparing final query quality we use the relative (scaled) error $\epsilon' = \varepsilon / |Q|$ [16] where $|Q| = \sum_{\rho_i \in Q} 1$.

When accumulating scalar rank or frequency estimates directly using Equation 2 the error $\varepsilon_Q(x)$ is just the sum of the errors introduced by the segment summaries for Q:

$$\varepsilon_Q(x) = \sum_{\mathcal{D}_i \in Q} \varepsilon_{\mathcal{D}_i}(x) = \sum_{\mathcal{D}_i \in Q} \left(g_{\mathcal{D}_i}(x) - g_{S_i}(x)\right) \tag{3}$$

When using the accumulator A a quantile or heavy hitter estimate will be based off the distribution defined by the $\hat{g}_S(x)$, but A introduces its own additional error $\varepsilon_S^{(A)}$ in approximating the proxy values in the summaries S, yielding a total error of:

$$\varepsilon_Q^{(A)}(x) \leq |\varepsilon_Q(x)| + |\varepsilon_S^{(A)}(x)|. \tag{4}$$

Furthermore we are interested in systems that provide error bounds over all values of x, so we consider the worst case error $\varepsilon_Q^{(A)} \coloneqq \max_x |\varepsilon_Q^{(A)}(x)|$. 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 Q accumulating k segments, each with total weight $n = |\mathcal{D}| = \sum_{x_i \in \mathcal{D}} \delta_i$ and represented using summaries of size s. Also, suppose that the accumulator A has size $s_A \gg s$. Suppressing logarithmic factors, state of the art frequency and quantile summaries have absolute error $O(n/s)$ [5, 16, 30, 40]. Different summarization techniques yield different errors as the size of the aggregation k grows. Storyboard can reduce relative error significantly for large k. We summarize the error bounds in Table 1.

Using mergeable summaries [5] for both S and the accumulator A gives us maximum absolute error $\varepsilon_Q^{(\text{merge})} \leq O(|Q|/s) \leq O(kn/s)$ and maximum relative error

$$\epsilon'^{(\text{merge})}_Q \leq O(1/s). \tag{5}$$

Storyboard's accumulator A applied to standard summaries gives us in the worst case $\varepsilon_Q^{(\text{A})\text{naive}} \leq \sum_{\mathcal{D}_i \in Q} |O(n/s)| +$

Table 1: Summary Error ignoring constants combining k summaries. The summaries used by storyboard: `CoopQuant`, `CoopFreq`, and PPS, all have reduced errors as for large k.

Summary	Relative ϵ'_Q	Tot. Space
`CoopFreq`	$\log k_T/(sk) + 1/s_A$	sk
`CoopQuant`	$\sqrt{k_T}/(sk) + 1/s_A$	sk
PPS [14]	$1/(s\sqrt{k}) + 1/s_A$	sk
Mergeable [5]	$1/s$	sk
Uniform Sample	$1/\sqrt{sk} + 1/s_A$	sk
Hierarchical [8, 41]	$\log k/(sk) + 1/s_A$	$sk \log k_T$

$O(nk/s_A)$ so

$$\epsilon'^{(\text{A})\text{naive}}_{Q} \leq O(1/s) + O(1/s_A) \tag{6}$$

where the $O(1/s_A)$ is negligible for $s_A \gg s$.

However, Storyboard 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 5.1 – 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 $\sum_{\mathcal{D}_i \in Q} \varepsilon_{\mathcal{D}_i}(x) \leq O(\sqrt{k}n/s)$ so

$$\epsilon'^{(\text{A})\text{PPS}}_{Q} \leq O\left(\frac{1}{\sqrt{ks}}\right) + O(1/s_A). \tag{7}$$

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

In practice, Cooperative summaries (Section 4) achieve even better error than PPS summaries for interval queries. We can prove that cooperative quantile summaries satisfy $\max_x |\varepsilon^{(A)\texttt{CoopQuant}}_{Q}(x)| \leq O(n\sqrt{k_T}/s)$

$$\epsilon'^{(A)\texttt{CoopQuant}}_{Q} \leq O\left(\frac{\min(\sqrt{k_T}, k)}{ks}\right) + O(1/s_A) \tag{8}$$

with much better empirical performance over intervals than PPS, while cooperative frequency summaries satisfy $\max_x |\varepsilon^{(A)\texttt{CoopFreq}}_{Q}(x)| \leq O(n \log k_T/s)$

$$\epsilon'^{(A)\texttt{CoopFreq}}_{Q} \leq O\left(\frac{\min(\log k_T, k)}{ks}\right) + O(1/s_A) \tag{9}$$

where k_T is the maximum length of an interval. See Section 4.2 for more details and proof sketches.

Hierarchical estimation is a common solution for interval (range) queries [8, 18] and show up in differential privacy as well [17]. 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 $\varepsilon^{(\text{A})\text{Hier}}_{Q} \leq O(n \log k/s) + O(1/s_A)$, 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 our Cooperative summaries.

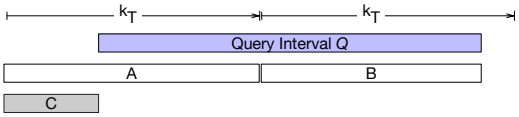


Figure 4: Any contiguous interval can be expressed as a linear combination of aligned intervals Pre_t. In this example, Q is expressed as $A \cup B \setminus C$

4 COOPERATIVE SUMMARIES

In this section we describe the cooperative summarization algorithms Storyboard uses for interval queries. These summaries achieve high query accuracy when combined by compensating for accumulated errors over sequences of summaries.

4.1 Interval Summary Construction

For interval queries, we assume that users specify a space limit s to use for summarizing an incoming data segment $\mathcal{D}$. Cooperative summaries then must make efficient use of their s samples $(x_i \mapsto y_i)$ to accurately represent a local segment of data. To match state of the art summaries, we want

$$\max_x |\hat{g}_S(x) - g_{\mathcal{D}}(x)| \leq r|\mathcal{D}|/s \tag{10}$$

for an accuracy parameter $r \geq 1$. However, there are many possible ways to choose the items to store in S that would satisfy Equation 10.

Within these constraints, Storyboard can choose x_i, y_i to minimize the total error for queries that aggregate multiple summaries. Storyboard explicitly minimizes the error over a set of queries with fixed start points every k_T segments. We call these aggregation intervals "prefix" intervals Pre_t, a modification of standard prefix-sum ranges [27].

$$\text{Pre}_t = \{\mathcal{D}_{k_T\lfloor t/k_T \rfloor}, \ldots, \mathcal{D}_{k_T\lfloor t/k_T \rfloor + t \bmod k_T}\}. \tag{11}$$

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 $\varepsilon_{\text{Pre}_t} = g_{\text{Pre}_t} - \hat{g}_{\text{Pre}_t}$, any contiguous interval up to length k_T has error at most 3ε. In order to minimize $\varepsilon_{\text{Pre}_t}(x)$ Storyboard will have to track its exact values, which may be resource intensive but is done during data ingest. The details of the summary construction algorithm differ for frequencies and ranks.

In Algorithm 1 we present pseudocode for constructing a cooperative summary of size s for frequency estimates on a data segment $\mathcal{D}_t$. To satisfy Equation 10 and accurately represent $\mathcal{D}_t$, we store the true count for any segment-local

Algorithm 1 Cooperative Item Frequencies Summary

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

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 compensating the x with the highest cumulative undercount $\varepsilon_{\text{Pre}_t}(x)$ thus far so that overcounting x in S_t will adjust for the undercount in the other summaries going forward. For each of these compensating x, we store the smaller of $r|\mathcal{D}|/s$ and $\varepsilon_{\text{Pre}_t}(x)$. This ensures Equation 10 is satisfied and also keeps $\varepsilon_{\text{Pre}_t}(x)$ 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 Quantile Summary

function CoopQuant($\mathcal{D}_t, s$)
  $h \leftarrow |\mathcal{D}_t|/s$; $S_t \leftarrow \{\}$
  $\varepsilon_{\text{Pre}_t}(x) \leftarrow \varepsilon_{\text{Pre}_{t-1}}(x) + r_{\mathcal{D}_t}(x)$
  $\mathcal{D}_{t1}, \ldots, \mathcal{D}_{ts} \leftarrow$ Partition($\mathcal{D}_t, s$) ▷ Sorted Chunks
  for $i \in 1 \ldots s$ **do**
    $L(z) \coloneqq \sum_{y \in U} \phi(\varepsilon_{\text{Pre}_t}(y))$
    $x_s \leftarrow \arg\min_{z \in \mathcal{D}_{ti}} L(z)$ ▷ Minimize Loss
    $S_t \leftarrow S_t \cup \{x_s \mapsto h\}$
    $\varepsilon_{\text{Pre}_t}(x) \leftarrow \varepsilon_{\text{Pre}_t}(x) - h \cdot 1_{x \geq x_s}$
  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 10 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 the 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 $L = \sum_{x \in U} \phi(\varepsilon_{\text{Pre}_t}(x))$ 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 [44] 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 $n_{\max}$

4.2 Interval Query Error

`CoopFreq` and `CoopQuant` both provide estimates with local error $\varepsilon_{\mathcal{D}}(x) \leq r|\mathcal{D}|/s$ for a single segment $\mathcal{D}$, and minimize the cumulative error over $\varepsilon_{\text{Pre}_t}(x)$ prefix intervals (and thus general intervals). In this section we analyze how $\varepsilon_{\text{Pre}_t}(x)$ grows with t. This allows us to prove Equations 8 and 9 in Section 3.4 which bound the query error from accumulating results over any sequence of k_T summaries.

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

$$L_t \coloneqq \sum_{x \in U} \phi\left(\varepsilon_{\text{Pre}_t}(x)\right) \tag{12}$$

where U is the universe of observed values $x \in |\text{Pre}_t|$. 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 $\max_x |\varepsilon_{\text{Pre}_t}(x)|$ to bound the latter. Omitted proofs in this section can be found in Appendix A.

4.2.1 `CoopFreq` *Error.* For frequency summaries, we use the cost function $\phi(x) = \exp(\alpha x)$ for a parameter α. We minimize a sum of exponentials 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, we can bound 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 $\alpha = 2\frac{s}{\max_i |\mathcal{D}_i|}\frac{r-1}{r^2}$.

Proof. α satisfies the conditions in Lemma 1 so $L_t \leq L_0 + \alpha r \sum_{i=0}^{t} |\mathcal{D}_i|$ and thus

$$\exp(\alpha \max_{x \in U} \varepsilon_t(x)) - 1 \leq \alpha r \sum_{i=0}^{t} |\mathcal{D}_i|$$

□

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 Corrollarry 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.

4.2.2 `CoopQuant` *Error.* For rank queries we use the cost function $\phi(x) = \cosh \alpha x$. 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 of the error, also used in discrepancy theory [44]. As with `CoopFreq`, Lemma 2 bounds how much L_t can increase with 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 $\phi(x) = \cosh(\alpha x)$

From Lemma 2 we can bound the maximum rank 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 $\alpha = s\left(\sum_{i=0}^t |\mathcal{D}_i|^2\right)^{-1/2}$.

Proof. Using Lemma 2:

$$L_t \leq L_0 \exp\left(\alpha^2/2 \sum_{i=0}^t \left(\frac{|\mathcal{D}_i|}{s}\right)^2\right)$$

$$\max_{x \in U} |\varepsilon_t(x)| \leq \frac{1}{\alpha}\ln(2|U|) + \frac{\alpha}{2}\sum_{i=0}^t \left(\frac{|\mathcal{D}_i|}{s}\right)^2$$

Then setting $\alpha = s/\sqrt{\sum_{i=0}^t |\mathcal{D}_i|^2}$ completes the proof. □

This can be instantiated for data segments with constant total weight in Corrollary 2, which shows that `CoopQuant` has 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*

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

5 OPTIMIZING CUBE QUERIES

In this section we describe the weighted probability proportional to size samples (PPS) [14, 45] Storyboard uses to summarize segments for both frequency and rank data cube aggregations. These randomized summaries have errors which cancel out with high probability. Then we describe how Storyboard optimizes the allocation of space and bias to these summaries to increase average query accuracy further for target cube workloads.

5.1 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 $\mathcal{D}$ [14, 45, 49]. Values x_i with true occurence count $\mathcal{D}(x_i) = \delta_i$ are sampled for inclusion in the summary S according to Equation 13

$$\Pr[x_i \in S] = \min(1, \delta_i/h) \tag{13}$$

$$S(x_i) = \begin{cases} h & \delta_i \leq h \\ \delta_i & \delta_i > h \end{cases}. \tag{14}$$

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 weight of h or excluded from the summary. Thus, $S(x_i)$ is an unbiased estimate for δ_i with maximum local error of h. We will see later that rank estimates $\hat{r}_S(x)$ $\sum_{x_j \in S} \gamma_j \cdot 1_{x_j \leq x}$ can also bounded by h.

Setting $h \leq |\mathcal{D}|/s$ ensures that the summary will have expected size s, but is conservative. In Algorithm 3 we present the procedure from [14] we use to set h to minimize error while keeping the summary size at most s by excluding the effect of heavy hitters.

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 13, 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 [14] 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
```

5.2 Cube Summary Construction

For data cubes, storyboard maintains a collection of PPS summaries for data segments $\mathcal{D}_i$ that form a complete, atomic partition of combinations of dimension values. A cube query then specifies a set of segments $Q_s = \{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$ that match dimension value filters. Storyboard ingests a complete cube dataset in batch and is given a total space budget S_T for storing summaries. 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, in Storyboard we make use of limited space by optimizing for the average error of queries sampled from a probabilistic workload W specified by the user. We show in Section 6.3.1 that the workload does not have to be perfectly specified to achieve accuracy improvements.

We consider a workload W as a distribution over possible queries Q_i where $\Pr[Q_i \sim W] = q_i$. This is based off of the workloads in STRAT [12], 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 $\epsilon' \leq \frac{1}{s_{\min}}$.

5.3 Minimizing Average Error

Consider the error incurred by combining summaries over a query $Q = \mathcal{D}_1, \ldots, \mathcal{D}_k$, 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 13, the relative error $\epsilon'_Q(x)$ is a random variable that depends on the items selected for inclusion in the PPS summaries. We will bound the mean squared relative 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:

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

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

$$E_{Q_z \sim W}\left[\epsilon'^2_Q\right] \leq \frac{1}{4} \sum_{\mathcal{D}_i \in \mathcal{D}} \frac{n_i^2}{s_i^2} \left(\sum_{z | \mathcal{D}_i \in Q_z} q_z |Q_z|^{-2} \right) \tag{15}$$

We can solve for the s_i that minimize the RHS of Equation 15 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

$$\alpha_i = n_i^2 \sum_{z | \mathcal{D}_i \in Q_z} q_z |Q_z|^{-2} \tag{16}$$

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 PPS 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 $n^2(\frac{1}{n} \cdot (1 - \frac{1}{n})) = n(1 - \frac{1}{n}) > 3$.

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

$$n_i[b] = \sum_{x_i \in \mathcal{D}} (\delta_i - b)^+ \tag{17}$$

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

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

query Q is:

$$E\left[\epsilon'^2_Q\right] \leq |Q|^{-2}\left(\left(\sum_{\mathcal{D}_i \in Q} b_i\right)^2 + \sum_{\mathcal{D}_i \in Q} \frac{1}{4}\left(\frac{n_i\,[b_i]^2}{s_i^2}\right)\right) \quad (18)$$

Equation 17 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 18 is convex as well.

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

(1) Set summary sizes $s_i \propto \alpha_i^{1/3}$ using Equation 16, scaled so $\sum s_i = S_T$.
(2) Solve for biases $\vec{b}$ that minimize the RHS of Equation 18 for Q a query over the entire dataset.
(3) Construct PPS summaries according to $\vec{s}$ and $\vec{b}$

We optimize Equation 18 using the LBFGS-B solver [10] in SciPy [29]. 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 relative error over any other query compared to $\vec{b} = 0$. Finding efficient and accurate proxies to optimize is a direction for future work.

6 EVALUATION

In our evalution, we show that:

(1) Storyboard's cooperative summaries achieve lower error as interval length increases compared with other summarization techniques: up to 8× for frequencies and 25× for quantiles (Section 6.2.1).
(2) Storyboard's space and bias optimizers provide lower average error for cube queries compared with alternative techniques, with reductions between 15% to 4.4× (Section 6.2.2).
(3) Storyboard's accuracy generalizes across different system and summary parameters, including accumulator size, maximum interval length, and workload specification (Sections 6.3.1 and 6.3.2).

6.1 Experimental Setup

Error Measurement. Recall from Section 3.4 that we are interested in error bounds that are independent of a specific item or value x, so we look at the maximum error over values x, $\epsilon'_Q := \max_x |\epsilon'_Q(x)|$. For a large domain of values U it is infeasible to compute $\max_{x \in U}$ so for frequency queries we estimate this maximum over a sample of 200 items drawn at random from the data excluding duplicates and for rank queries over a set of 200 equally spaced values from the global value distribution. Following common practice for approximate summaries [16], we rescale the absolute error by the total size of the queried data to report relative errors $\epsilon'_Q = \varepsilon_Q/|Q|$.

The final query error $\epsilon'^{(A)}_Q$ is then bounded by the sum of the summary and accumulator errors $\epsilon'_Q + \epsilon'^{(A)}$. In our evaluations we assume that the accumulator is large enough to introduce negligible additional error, and confirm that the additional error rapidly vanishes as the size of the accumulator grows in Figure 7. When summaries provide a native merge routine, we still benchmark query accuracy by accumulating their estimates rather than merging the summaries directly. This is strictly more accurate than merging, which will further compress intermediate results to fit in the original summary space, and provides a more fair comparison with Storyboard.

Implementation. We evaluate a prototype implementation of Storyboard written in Python with core summarization and query processing logic compiled to C and code available[1]. Since our focus is query accuracy under space constraints, 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` in Algorithm 3 rather than letting $h = |\mathcal{D}_t|/s$. $h :=$ `CalcT` gives us better segment accuracy and the error bounds still hold 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.

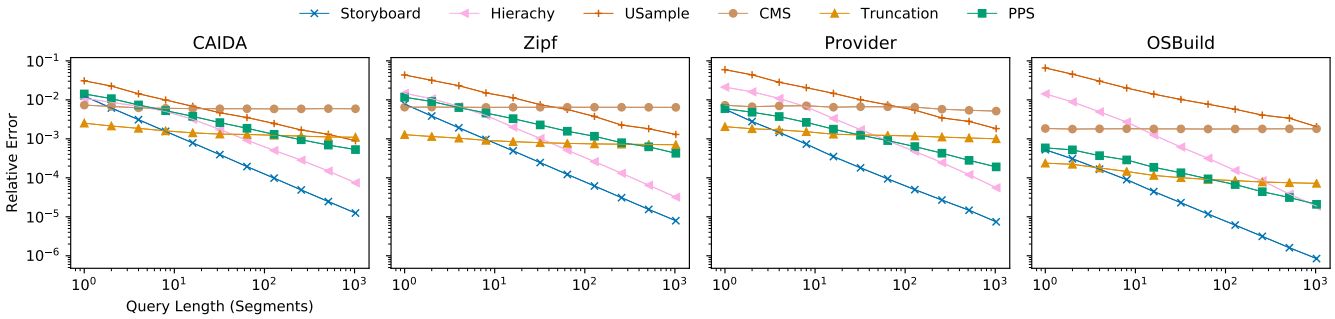
Datasets. We evaluate frequency estimates on 10 million destination ip addresses (`CAIDA`) from a Chicago Equinix backbone on 2016-01-21 available from CAIDA [11], 10 million items (`Zipf`) drawn from a Zipf (Pareto) distribution with parameter $s = 1.1$, and 10 million records from a production service 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 [20], 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`).

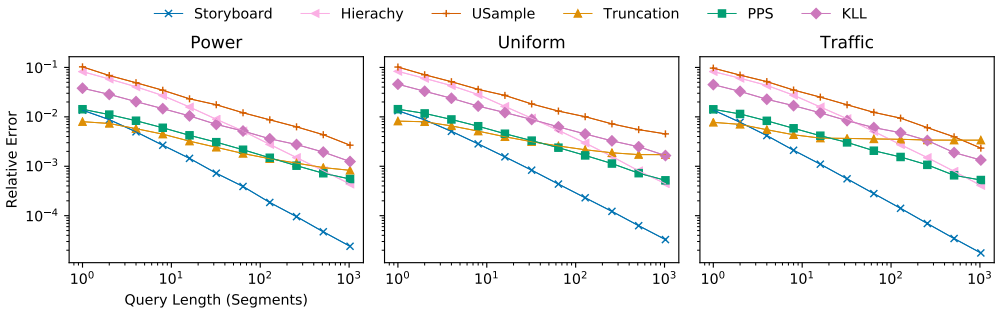
6.2 Overall Query Accuracy

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

[1] https://github.com/stanford-futuredata/sketchstore



(a) Frequency Queries



(b) Quantile Queries

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

accuracy. For all counter and sample-based summaries including `CoopFreq`, `CoopQuant`, and PPS, we set the number of counters or samples to the same s.

We compare against two popular mergeable summaries: the optimal quantiles sketch (`KLL`) from [30], and the Count-Min frequency sketch (`CMS`) [18]. 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`) [15], and optimal single-segment summaries (`Truncation`) 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 also compare with storing `Truncation` summaries in a hierarchy (`Hierarchy`) following [8, 17]. Specifically, 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 6.3 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 `USample:Prop` method uses `USample` summaries but allocates space proportional to each segment size as a baseline random reservoir sample would [46] while the `STRAT` method uses the method in the STRAT AQP system [12], which like Storyboard allocates space to minimize average error.

6.2.1 Interval Queries. We first evaluate Storyboard 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 100 random start and end times for intervals with length k and plot the average and standard deviation of the query error.

Storyboard, which uses cooperative summaries, outperforms any system that uses mergeable summaries or random sampling as k increases. As the interval length increases merging the mergeable summaries (`CMS`, `KLL`) maintain their

Table 2: Cube Datasets

Data	# Segments	Summary Space
Instacart	10080	300000
Zipf, Uniform	10000	50000
Traffic	4613	50000
OSBuild, Provider	4613	100000

error as expected. Accumulating Truncation 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 also observe that despite our weaker worst-case bounds for cooperative quantile summaries, they achieve higher accuracy in practice compared to alternative methods. However, Storyboard gives up a constant factor in accuracy when aggregating less than 10 summaries compared to alternatives.

6.2.2 Cube Queries. We evaluate cube queries on our datasets with categorical dimension columns, and partitioned them along four dimension columns with parameters summarized in Table 2 and total space limit set to provide roughly consistent query error across the datasets. For each of these 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 over the different dataset workloads. We see that, on average, Storyboard outperforms alternative summarization techniques that allocate equal space to each segment, as well as uniform sampling techniques that optimize sample size allocation.

6.3 Varying Parameters

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

6.3.1 System Design. The Storyboard system depends on a number of parameters that go beyond a single summary. In this section we will show how accuracy varies with the accumulator size s_A, the size of cube aggregations, and the presence of each of the optimizations storyboard uses for data cubes.

Finite Accumulator. As described in Section 3.1, Storyboard accumulates precise frequency and ranks estimates from the summaries for point queries $\hat{g}(x)$, and accumulates summaries into a large accumulator A for quantile and heavy hitters queries. In our evaluations thus far we have measured the maximum point query error, which bounds the maximum quantile or heavy hitter error as well (Section 3.4). The accumulator A introduces an additional approximation error $\epsilon'^{(A)} = 1/s_A$ which is negligible as $s_A \to \infty$.

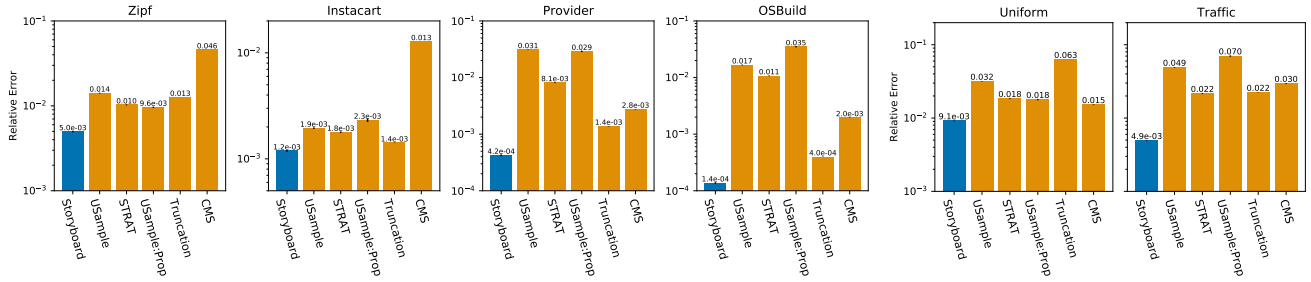
In Figure 7 we illustrate how using accumulators of different sizes, affects final query accuracy on the Power and CAIDA datasets. For each accumulator size, we measure the error after accumulating 100 random interval aggregations spanning $k = 512$ segments. For the accumulators here we use SpaceSaving [34] for frequency queries and a streaming implementation of PPS (VarOpt [14]) for quantiles. $\epsilon'^{(A)}$ quickly goes to 0 as $s_A \to \infty$, so with at least 10 megabytes of memory available for s_A the additional error is negligible.

Cube Query Spans. As seen already for interval queries, Storyboard improves the query error for queries combining results from multiple segment summaries. We can confirm this for cubes by looking at query error broken down by the number of dimensions in each query filter condition. Queries that filter on fewer columns will combine results from more segments. In Figure 8 we compare the error for queries that filter on different numbers of dimensions on the Uniform and Zipf cube workloads. Storyboard reduces the error for common queries that filter on zero or one dimension. As a tradeoff Storyboard incurs higher error than other methods for rarer queries with three or more filters. For a workload where queries with 3 or more filters are much less common than queries with 0 or 1 filters, this tradeoff is useful, and is configurable based on the user specified workload.

Cube Optimizer Lesion Study. In Figure 9 we show how the optimizations Storyboard (SB) uses 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 (SB (-Size)) and bias optimizations (SB (-Bias)), and try replacing PPS summaries with uniform random samples (SB (-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 Storyboard accuracy depends on precise workload specification by constructing Storyboard instances configured for incorrectly specified workloads. Rather than the true $p = 0.2$ probability of including a dimension in the cube filter, we try optimizing cubes for $p = 0.05$ in Work1 and $p = 0.50$ in Work2. As included in Figure 9, in both cases error remains below existing cube construction methods. Interestingly, accuracy improves for Work1, indicating our optimization is not tight.

Interval Length Specification. For interval aggregations users specify a maximum expected interval length k_T. In



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

Figure 6: Average query error over a workload of cube queries.

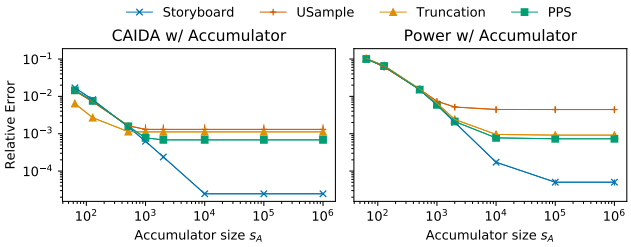


Figure 7: Query error as we vary the size of the accumulator s_A. For the large accumulators used in practice there is negligible additional error from the accumulator.

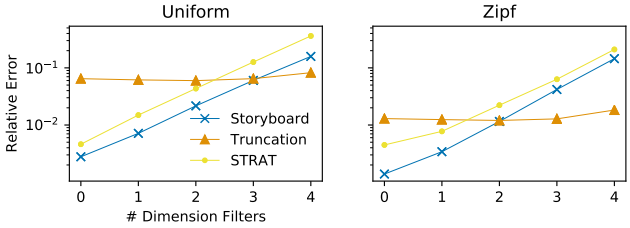


Figure 8: Query error broken down by number of dimension filters in a query. Storyboard achieves lower error on queries that have fewer filters and aggregate more segments.

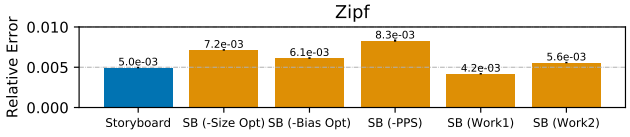


Figure 9: Lesion Study on `Zipf` cube optimizations. Removing any component reduces accuracy, though adjusting the workload parameter slightly improves accuracy.

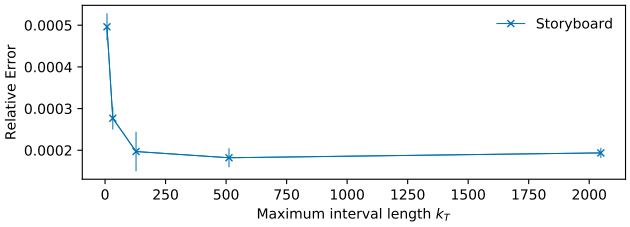


Figure 10: Error as we vary the maximum interval length parameter k_T. Overestimating k_T does not significantly change the quality of results.

Figure 10 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 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.

6.3.2 Summary Design. Now we will examine how Storyboard's Cooperative and PPS summaries perform as individual segment summaries. The experiments below are run on the `CAIDA` dataset for interval aggregations.

Space Scaling. In Figure 11 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 12 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.

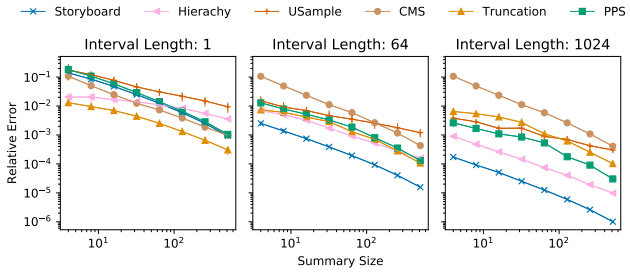


Figure 11: Query error as summary size changes. Cooperative summaries, like state of the art, have error $\epsilon' = O(1/s)$

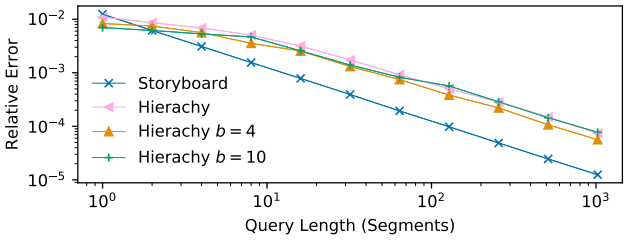


Figure 12: `Hierarchy` summary accuracy for different bases b. b does not have a large impact when accumulating across multiple summaries.

.

7 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 [31], and they are an instance of the AggPre systems described in [38]. Like data cube systems they materialize partial results [25], but can support more complex query functions not captured by simple totals. Another class of systems use "online" AQP [9, 26, 42] and provide different latency and accuracy guarantees by computing approximations at query-time.

We are particularly motivated by Druid [43, 47] and similar offline systems [28] 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 [48] 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 [6], STRAT [12], and AQUA [4] maintain random stratified samples to support general-purpose queries. Our choice of minimizing mean squared error over a workload follows the setup in STRAT [12]. However, individual simple random samples are not as accurate as specialized frequency or quantile summaries [36].

Techniques for summarizing hierarchical intervals [8] are complementary, but incur additional storage overhead making them less accurate than Cooperative summaries and scale poorly to cubes with multiple dimensions [41].

Streaming and Mergeable Summaries. Many compact data summaries are developed in the streaming literature [18, 24, 30, 35], including summaries for sliding windows [7]. However, they assume a different system model than Storyboard provides. The standard streaming model generally considers queries with working memory limited during summary construction [37]. Mergeable summaries [5] allow combining multiple summaries but require that intermediate results take up 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 [22, 33] rather than improving the accuracy of merged summary results.

Other Summarization Models. The Storyboard 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 [40]. ϵ-approximations are part of discrepancy theory which attempts to approximate an underlying distribution with proxy samples [13]. We draw inspiration from discrepancy theory to manage error accumulation in our cooperative summaries, especially the results in [44] which pioneered the use of the cosh cost function. Other work in this area minimize error accumulation along multiple dimensions [39]. However, we are not away of coreset or ϵ-approximations that allow for complex queries Storyboard supports: quantiles and item frequencies over multiple data segments. and cube aggregations. In particular, existing work supporting range queries [39] do not provide per-segment local guarantees.

Work in communication theory and distributed streaming assume a setting where sending summaries over the network is a bottleneck equivalent to storage costs limits in Storyboard. There is existing work analyzing how multiple random samples can be combined to reduce aggregate error in this setting [49, 50]. However, in communication theory the samples are constructed per-query, while Storyboard precomputes summaries that can be used for arbitrary future queries. Furthermore the random samples are not as space efficient as cooperative summaries.

Related techniques in differential privacy [17, 41] and matrix rounding [19] 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.

8 CONCLUSION

When aggregating multiple precomputed summaries, Storyboard optimizes and accumulates summaries for reduced query error. It does so by taking advantage of additional memory resources at summary construction and aggregation while targeting a common class of structured frequency and quantile queries. This system can thus efficiently serve a range of monitoring and data exploration workloads. Extensions to other query and aggregation types are a rich area for future work.

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. We also thank members of the Stanford Infolab for their feedback on early drafts of this paper. 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, Amazon Web Services, Cisco, and the NSF under CAREER grant CNS-1651570.

REFERENCES

[1] 2020. DataSketches Quantiles Sketch module. https://druid.apache.org/docs/latest/development/extensions-core/datasketches-quantiles.html.

[2] Firas Abuzaid, Peter Bailis, Jialin Ding, Edward Gan, Samuel Madden, Deepak Narayanan, Kexin Rong, and Sahaana Suri. 2018. MacroBase: Prioritizing Attention in Fast Data. *ACM Trans. Database Syst.* 43, 4, Article 15 (2018), 45 pages.

[3] Firas Abuzaid, Peter Kraft, Sahaana Suri, Edward Gan, Eric Xu, Atul Shenoy, Asvin Ananthanarayan, John Sheu, Erik Meijer, Xi Wu, Jeff Naughton, Peter Bailis, and Matei Zaharia. 2018. DIFF: A Relational Interface for Large-scale Data Explanation. *PVLDB* 12, 4 (2018), 419–432.

[4] Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. 1999. The Aqua Approximate Query Answering System. In *SIGMOD*. 574–576.

[5] Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff Phillips, Zhewei Wei, and Ke Yi. 2012. Mergeable Summaries. In *PODS*.

[6] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *EuroSys*. 29–42.

[7] Arvind Arasu and Gurmeet Singh Manku. 2004. Approximate Counts and Quantiles over Sliding Windows. In *PODS*. 286–296.

[8] Ran Ben Basat, Roy Friedman, and Rana Shahout. 2018. Stream Frequency over Interval Queries. *PVLDB* 12, 4 (2018), 433–445.

[9] Mihai Budiu, Parikshit Gopalan, Lalith Suresh, Udi Wieder, Han Kruiger, and Marcos K. Aguilera. 2019. Hillview: A Trillion-cell Spreadsheet for Big Data. *PVLDB* 12, 11 (July 2019), 1442–1457.

[10] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. 1995. A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM J. Sci. Comput.* 16, 5 (1995), 1190–1208.

[11] CAIDA. 2016. The CAIDA UCSD Anonymized Internet Traces. http://www.caida.org/data/passive/passive_dataset.xml http://www.caida.org/data/passive/passive_dataset.xml.

[12] Surajit Chaudhuri, Gautam Das, and Vivek Narasayya. 2007. Optimized Stratified Sampling for Approximate Query Processing. *ACM Trans. Database Syst.* 32, 2 (2007).

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

[14] Edith Cohen, Graham Cormode, and Nick G. Duffield. 2011. Structure-Aware Sampling: Flexible and Accurate Summarization. *PVLDB* 4, 11 (2011), 819–830.

[15] Graham Cormode, Minos Garofalakis, Peter J Haas, and Chris Jermaine. 2012. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases* 4, 1–3 (2012), 1–294.

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

[17] Graham Cormode, Tejas Kulkarni, and Divesh Srivastava. 2019. Answering Range Queries Under Local Differential Privacy. *PVLDB* 12, 10 (2019), 1126–1138.

[18] Graham Cormode and S. Muthukrishnan. 2005. An Improved Data Stream Summary: The Count-min Sketch and Its Applications. *J. Algorithms* 55, 1 (2005), 58–75.

[19] Benjamin Doerr, Tobias Friedrich, Christian Klein, and Ralf Osbild. 2006. Unbiased Matrix Rounding. In *Algorithm Theory – SWAT 2006*, Lars Arge and Rusins Freivalds (Eds.). 102–112.

[20] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[21] Ted Dunning and Otmar Ertl. 2019. Computing extremely accurate quantiles using t-digests. https://github.com/tdunning/t-digest. *arXiv preprint arXiv:1902.04023* (2019).

[22] Edward Gan, Jialin Ding, Kai Sheng Tai, Vatsal Sharan, and Peter Bailis. 2018. Moment-based Quantile Sketches for Efficient High Cardinality Aggregation Queries. *PVLDB* 11, 11 (July 2018), 1647–1660.

[23] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. 1997. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery* 1, 1 (1997), 29–53.

[24] Michael Greenwald and Sanjeev Khanna. 2001. Space-efficient online computation of quantile summaries. In *SIGMOD*, Vol. 30. 58–66.

[25] Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. 1996. Implementing Data Cubes Efficiently. In *SIGMOD*. 205–216.

[26] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. 1997. Online Aggregation. In *SIGMOD*. 171–182.

[27] Ching-Tien Ho, Rakesh Agrawal, Nimrod Megiddo, and Ramakrishnan Srikant. 1997. Range Queries in OLAP Data Cubes. *SIGMOD* 26, 2 (1997), 73–88.

[28] Jean-François Im, Kishore Gopalakrishna, Subbu Subramaniam, Mayank Shrivastava, Adwait Tumbde, Xiaotian Jiang, Jennifer Dai, Seunghyun Lee, Neha Pawar, Jialiang Li, and Ravi Aringunram. 2018. Pinot: Realtime OLAP for 530 Million Users. In *SIGMOD*. 583–594.

[29] Eric Jones, Travis Oliphant, Pearu Peterson, et al. 2001–. SciPy: Open source scientific tools for Python. http://www.scipy.org/ [Online; accessed Oct 2019].

[30] Z. Karnin, K. Lang, and E. Liberty. 2016. Optimal Quantile Approximation in Streams. In *FOCS*. 71–78.

[31] Kaiyu Li and Guoliang Li. 2018. Approximate Query Processing: What is New and Where to Go? *Data Science and Engineering* 3, 4 (01 Dec 2018), 379–397.

[32] Ge Luo, Lu Wang, Ke Yi, and Graham Cormode. 2016. Quantiles over data streams: experimental comparisons, new analyses, and further improvements. *VLDB* 25, 4 (2016), 449–472.

[33] Charles Masson, Jee E. Rim, and Homin K. Lee. 2019. DDSketch: A Fast and Fully-Mergeable Quantile Sketch with Relative-Error Guarantees. *PVLDB* 12, 12 (Aug. 2019), 2195–2205.

[34] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2005. Efficient Computation of Frequent and Top-k Elements in Data Streams. In *Proceedings of the 10th International Conference on Database Theory (ICDT'05)*. 398–412.

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

[36] Barzan Mozafari and Ning Niu. 2015. A Handbook for Building an Approximate Query Engine. *IEEE Data Eng. Bull.* 38, 3 (2015), 3–29.

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

[38] Jinglin Peng, Dongxiang Zhang, Jiannan Wang, and Jian Pei. 2018. AQP++: Connecting Approximate Query Processing With Aggregate Precomputation for Interactive Analytics. In *SIGMOD*. 1477–1492.

[39] Jeff M. Phillips. 2008. Algorithms for ϵ-Approximations of Terrains. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming - Volume Part I (ICALP '08)*. 447–458.

[40] Jeff M. Phillips. 2017. Coresets and Sketches. In *Handbook of Discrete and Computational Geometry*, Csaba D. Toth, Joseph O'Rourke, and Jacob E. Goodman (Eds.). CRC Press, Chapter 48, 1267–1286.

[41] Wahbeh Qardaji, Weining Yang, and Ninghui Li. 2013. Understanding Hierarchical Methods for Differentially Private Histograms. *PVLDB* 6, 14 (Sept. 2013), 1954–1965.

[42] Ariel Rabkin, Matvey Arye, Siddhartha Sen, Vivek S. Pai, and Michael J. Freedman. 2014. Aggregation and Degradation in JetStream: Streaming Analytics in the Wide Area. In *NSDI*. 275–288.

[43] Nelson Ray. 2013. The Art of Approximating Distributions: Histograms and Quantiles at Scale. http://druid.io/blog/2013/09/12/the-art-of-approximating-distributions.html.

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

[45] Daniel Ting. 2018. Data Sketches for Disaggregated Subset Sum and Frequent Item Estimation. In *SIGMOD*. 1129–1140.

[46] Jeffrey S. Vitter. 1985. Random Sampling with a Reservoir. *ACM Trans. Math. Softw.* 11, 1 (1985), 37–57.

[47] Fangjin Yang, Eric Tschetter, Xavier Léauté, Nelson Ray, Gian Merlino, and Deep Ganguli. 2014. Druid: A Real-time Analytical Data Store. In *SIGMOD*. 157–168.

[48] Ke Yi, Lu Wang, and Zhewei Wei. 2014. Indexing for Summary Queries: Theory and Practice. *ACM Trans. Database Syst.* 39, 1 (Jan. 2014).

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

[50] Qi (George) Zhao, Mitsunori Ogihara, Haixun Wang, and Jun (Jim) Xu. 2006. Finding Global Icebergs over Distributed Data Sets. In *PODS*. 298–307.

A 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 $V = S_t \setminus H$.

$$S_H = \{x_i \mapsto \delta_i : x_i \in H\} \tag{19}$$

$$S_V = \{x_i \mapsto \min(\varepsilon_{t-1}(x_i) + \delta_i, rh) : x_i \in V\}. \tag{20}$$

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 $\phi(x+y) = \phi(x)\phi(y)$:

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

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

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

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 $\alpha \leq \frac{2}{h}\frac{r-1}{r^2}$, $G \leq \alpha r|\mathcal{D}_t|$ □

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

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

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. □

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

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

$$p\cosh(x+t(p-1)) + (1-p)\cosh(x+tp) \leq \exp(t^2/2)\cosh(x) \tag{21}$$

Proof. We abbreciate cosh, sinh as c, s and the left hand side of Equation 21 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:

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

□

A.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 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. □