Joins on Samples: A Theoretical Guide for Practitioners

Dawei Huang Dong Young Yoon Seth Pettie Barzan Mozafari
University of Michigan, Ann Arbor
{hdawei, dyoon, pettie, mozafari}@umich.edu

ABSTRACT

Despite decades of research on AQP (approximate query processing), our understanding of sample-based joins has remained limited and, to some extent, even superficial. The common belief in the community is that joining random samples is futile. This belief is largely based on an early result showing that the join of two *uniform* samples is not an independent sample of the original join, and that it leads to quadratically fewer output tuples. Unfortunately, this early result has little applicability to the key questions practitioners face. For example, the success metric is often the final approximation's accuracy, rather than output cardinality. Moreover, there are many non-uniform sampling strategies that one can employ. Is sampling for joins still futile in all of these settings? If not, what is the best sampling strategy in each case? To the best of our knowledge, there is no formal study answering these questions.

This paper aims to improve our understanding of sample-based joins and offer a guideline for practitioners building and using real-world AQP systems. We study limitations of offline samples in approximating join queries: given an offline sampling budget, how well can one approximate the join of two tables? We answer this question for two success metrics: output size and estimator variance. We show that maximizing output size is easy, while there is an information-theoretical lower bound on the lowest variance achievable by any sampling strategy. We then define a hybrid sampling scheme that captures all combinations of stratified, universe, and Bernoulli sampling, and show that this scheme with our optimal parameters achieves the theoretical lower bound within a constant factor. Since computing these optimal parameters requires shuffling statistics across the network, we also propose a decentralized variant in which each node acts autonomously using minimal statistics. We also empirically validate our findings on popular SQL and AQP engines.

PVLDB Reference Format:
Dawei Huang, Dong Young Yoon, Seth Pettie, and Barzan Mozafari. Joins on Samples: A Theoretical Guide for Practitioners. *PVLDB*, 13(4): 547-560, 2019.
DOI: https://doi.org/10.14778/3372716.3372726

1. INTRODUCTION

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 13, No. 4
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3372716.3372726

Approximate query processing (AQP) has regained significant attention in recent years due to major trends in the industry [43]. Larger datasets, memory wall, and the separation of compute and storage have all made it harder to achieve interactive-speed analytics. AQP presents itself as a viable alternative in scenarios where perfect decisions can be made with imperfect answers [8]. AQP is most appealing when negligible loss of accuracy can be traded for a significant gain in speedup or computational resources. Ad-hoc analytics [9, 56, 62], visualization [23, 49, 55], IoT [46], A/B testing [7], email marketing and customer segmentation [28], and real-time threat detection [2] are examples of such usecases.

Sampling and Joins— Sampling is one of the most widely-used techniques for general-purpose AQP [22]. The high level idea is to execute the query on a small sample of the original table(s) to provide a fast, but approximate, answer. While effective for simple aggregates, using samples for join queries has long remained an open problem [6]. There are two main approaches to AQP: *offline* or *online*. Offline approaches [5, 6, 8, 15, 27, 50] build samples (or other synopses) prior to query arrival. At run time, they simply choose appropriate samples that can yield the best accuracy/performance for each incoming query. Online approaches, on the other hand, wander-join perform much of their sampling at run time based on the query at hand [13, 20, 33, 37, 48, 59]. Naturally, offline sampling leads to significantly higher speedup, while online techniques can support a much wider class of queries [37]. The same taxonomy applies to join approximation: offline techniques perform joins on previously-prepared samples [6, 17, 18, 50, 63], while online approaches seek to produce a sample of the output of the join at run time [25, 29, 40, 42]. As mentioned, the latter often means more modest speedups (e.g., $2\times$ [37]) which may not be sufficient to justify approximation, or additional requirements (e.g., an index for each join column [40]) which may not be acceptable to many applications. Thus, our focus in this paper—and what is considered an open-problem—is the offline approach: joins on samples, not sampling the join's output.

Joins on Samples— The simplest strategy is as follows. Given two large tables T_1 and T_2, create a uniform random sample of each, say S_1 and S_2 respectively, and then use $S_1 \bowtie S_2$ to approximate aggregate statistics of $T_1 \bowtie T_2$. This will lead to significant speedup if samples are much smaller than original tables, i.e., $|T_i| \gg |S_i|$.

One of the earliest results in this area shows that this simple strategy is futile for two reasons [5]. First, joining two uniform samples leads to quadratically fewer output tuples, i.e., joining two uniform samples that are each p fraction ($0 \leq p < 1$) of the original tables will only produce p^2 of the output tuples of the original join (see Figure 1). Second, joining uniform samples of two tables does not

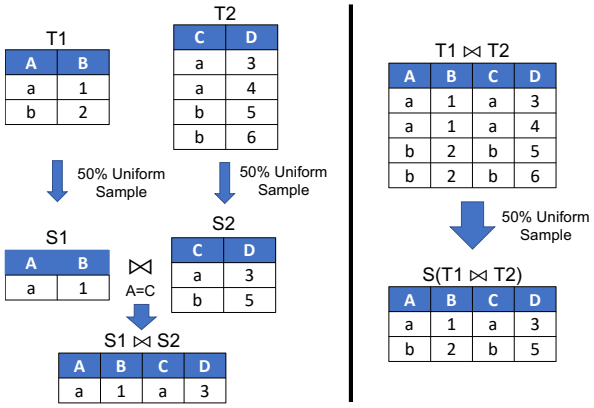


Figure 1: **A toy example of joining two uniform samples (left) versus a uniform sample of the join (right).**

yield an independent sample of their join[1] (see Section 2.1 for details). The dependence of the output tuples can drastically lower the approximation accuracy [5, 17].

Prior Work— *Universe* sampling [31, 37, 50] addresses the first drawback of uniform sampling. Although universe sampling avoids quadratic reduction of output, its creates even more correlation in its output, leading to much lower accuracy (see Section 3.1).

Atserias et al. provide a worst case lower bound for any query involving equi-joins on multiple relations, showing that computing exact joins with a small memory or time budget is hard [12]. For instance, the maximum possible join size for any cyclic join on three n-tuple relations is $\Theta(n^{1.5})$. Thus, a natural question is whether approximating joins is also hard with small memory or time.

Our Goal— This paper focuses on understanding the limitation of using offline samples in approximating join queries. Given a sampling budget, how well can we approximate the join of two tables using their offline samples? To answer this question, we must first define what constitutes a "good" approximation of a join. We consider two metrics: (1) output cardinality and (2) aggregation accuracy. The former is the number of tuples of the original join that also appear in the join of the samples, whereas the latter is the error of the aggregates estimated from the sample-based join with respect to their true values, if computed on the original join. Because in this paper we only consider unbiased estimators, we measure approximation error in terms of the variance of our estimators.

For the first metric, we provide a simple proof showing that universe sampling is optimal, i.e. no sampling scheme with the same sampling rate can outperform universe sampling in terms of the (expected) output cardinality. However, as we show in Section 3.1, retaining a large number of join tuples does not imply accurate aggregates. It is therefore natural to also ask about the lowest variance that can be achieved given a sampling rate. To the best of our knowledge, this has remained an open problem to date. For the first time, we formally study this problem and offer an information-theoretical lower bound to this question. We also present a hybrid sampling scheme that matches this lower bound within a constant factor. This scheme involves a centralized computation, which can become prohibitive for large tables due to large amounts of statistics that need to be shuffled across the network. Thus, we also propose a decentralized variant that only shuffles a minimal amount of information across the nodes—such as the table size and maximum frequency—but still achieves the same worst case guarantees. Finally, we generalize our sampling scheme to accommodate *a priori* information about filters (i.e., `WHERE` clause).

In this paper, we make the following contributions:

1. We discuss two metrics—output size and estimator's variance —for measuring the quality of join approximation, and show that universe sampling is optimal for output size and there is an information-theoretical lower bound for variance (Section 3).

2. We formalize a hybrid scheme, called Stratified-Universe-Bernoulli Sampling (SUBS), which allows for different combinations of stratified, universe, and Bernoulli sampling. We derive optimal sampling parameters within this scheme, and show that they achieve the theoretical lower bound of variance within a constant factor (Section 4–5.3). We also extend our analysis to accommodate additional information regarding the `WHERE` clause (Section 6).

3. Through extensive experiments, we also empirically show that our optimal sampling parameters achieve lower error than existing sampling schemes in both centralized and decentralized scenarios (Section 7).

2. BACKGROUND

In this section, we provide the necessary background on sampling-based join approximation. We also formally state our problem setting and assumptions.

2.1 Sampling in Databases

The following are the three main popular sampling strategies (operators) used in AQP engines and database systems.

1. **Uniform/Bernoulli Sampling.** Any strategy that samples all tuples with the same probability is considered a uniform (random) sample. Since enforcing fixed-size sampling without replacement is expensive in distributed systems, Bernoulli sampling is considered a more efficient strategy [37]. In Bernoulli sampling, each tuple is included in the sample independently, with a fixed sampling probability p. In this paper, for simplicity, we use "uniform" and "Bernoulli" interchangeably. As mentioned in Section 1, joining two uniform samples leads to quadratically fewer output tuples. Further, it does not guarantee an i.i.d. sample of the original join [6]: the output is a uniform sample of the join but not an independent one. Consider an arbitrary tuple of the join, say (t_1, t_2), where t_1 is from the first table and t_2 is from the second. The probability of this tuple appearing in the join of the samples is always the same value, i.e., p^2. The output is thus a uniform sample. However, the tuples are not independent: consider another tuple of the join, say (t_1, t_2') where t_2' is another tuple from the second table joining with t_1. If (t_1, t_2) appears in the output, the probability of (t_1, t_2') also appearing becomes p instead of p^2, which would be the probability if they were independent.

2. **Universe Sampling.** Given a column[2] J, a (perfect) hash function $h : J \mapsto [0, 1]$, and a sampling rate p, this strategy includes a tuple t in the table if $h(t.J) \leq p$. Universe sampling is often used for equi-joins, in which the same p value and hash function h are applied to the join columns in both tables. This ensures that when a tuple t_1 is sampled from one table, any matching

[1] Prior work has stated that joining uniform samples is not a *uniform* sample of the join [6]. We avoid this terminology since uniform means equal probability of inclusion, and in this case each tuple does appear in the join of the uniform samples with equal probability, but not independently. In other words, joining two i.i.d. samples is an identical, but not independent, sample of the join.

[2] J can also be a set of multiple columns.

tuple t_2 from the other table is also sampled, simply because $t_1.J = t_2.J \Leftrightarrow h(t_1.J) = h(t_2.J)$. This is why joining two universe samples of rate p produces p fraction of the original join output *in expectation*. The output is a uniform sample of the original join, as each join tuple appears with the same probability p. However, there is more dependence among the output tuples. Consider two join tuples (t_1, t_2) and (t'_1, t'_2) where t_1, t'_1, t_2, t'_2 all share the same join key. Then, if (t_1, t_2) appears, the probability of (t'_1, t'_2) also appearing will be 1. Likewise, if (t_1, t_2) does not appear, the probability of (t'_1, t'_2) appearing will be 0. Higher dependence means lower accuracy (see Section 3.1).

3. **Stratified Sampling.** The goal of stratified sampling is to ensure that minority groups are sufficiently represented in the sample. Groups are defined according to one or multiple columns, called the *stratified columns*. A group (a.k.a. a stratum) is a set of tuples that share the same value under those stratified columns. Given a set of stratified columns C and an integer parameter k_{tuple}, a stratified sampling is a scheme that guarantees at least k_{tuple} tuples are sampled uniformly at random from each group. When a group has fewer than k_{tuple} tuples, all of them are retained.

2.2 Quality Metrics

Different metrics can be used to assess the quality of a join approximation. In this paper, we focus on the following two, which are used by most AQP systems.

Output Size/Cardinality— This metric is the number of tuples of the original join that also appear in the join of the samples. It is mostly relevant for exploratory usecases, where users visualize or examine a subset of the output. In other cases, where an aggregate is computed from the join output, retaining a large number of output tuples does not guarantee accurate answers (we show this in Section 3.1).

Variance— In scenarios where an aggregate function needs to be calculated from the join output, the error of the aggregate approximation is more relevant than the number of intermediate tuples generated. For most non-extreme statistics, there are readily available unbiased estimators, e.g., Horvitz-Thompson estimator [34]. Thus, a popular indicator of accuracy is the variance of the estimator [8], which determines the size of the confidence interval given a sample size.

2.3 Problem Statement

In this section, we formally state the problem of sample-based join approximation. The notations used throughout the paper are listed in Table 1.

Query Estimator— Let S_1 and S_2 be two samples generated offline from tables T_1 and T_2, respectively, and q_{agg} be a query that computes an aggregate function agg on the join of T_1 and T_2. A query estimator $\hat{J}_{agg}(S_1, S_2)$ is a function that estimates the value of agg using two samples rather than the original tables.

Join Sampling Problem— Given a query estimator $\hat{J}_{agg}$ and a sampling budget $\epsilon \in (0, 1]$, our goal is to create a pair of samples S_1 and S_2—from tables T_1 and T_2, respectively— that are optimal in terms of a given success metric, while respecting a given storage budget $epsilon$ on average. Specifically, we seek S_1 and S_2 that minimize $\hat{J}_{agg}$'s variance or maximize its output size such that $E[|S_1| + |S_2|] \leq \epsilon \times (|T_1| + |T_2|)$.

Note that we define the sampling budget in terms of an expected size (rather than a strict one), since sampling schemes are probabilistic in nature and may slightly over- or under-use the budget.

Table 1: **Notations.**

Notation	Definition
T_1, T_2	Two tables for the join
S_i	A sample generated from table T_i
J	Column(s) used for the join between T_1 and T_2
W	Column being aggregated (e.g., `SUM`, `AVG`)
C	Column(s) used for filters (i.e., `WHERE` clause)
$\mathcal{U}$	Set of all possible values of J
a, b	Frequency vectors of T_1 and T_2's join columns, resp.
a_v, b_v	Number of tuples with join value v in T_1 and T_2, resp.
$\hat{J}_{agg}$	Estimator for a join query with aggregate function agg
ϵ	Sampling budget w.r.t. the original table size
n_1, n_2	Number of tuples in T_1 and T_2, resp.
h	A (perfect) hash function
k_{tuple}	Minimum number of tuples to be kept per group in stratified sampling
k_{key}	Minimum number of join keys per group to apply universe sampling (universe sampling is not applied to groups with fewer than k_{key} join keys)
p	Sampling rate of universe sampling
q	Sampling rate of uniform sampling

To formally study this problem, we first need to define a class of reasonable sampling strategies. In Section 4, we define a hybrid scheme that can capture different combinations of stratified, universe, and uniform sampling.

2.4 Scope and Limitations

To simplify our analysis, we limit our scope in this paper.

Flat Equi-joins— We focus on equi (inner) joins as the most common form of joins in practice. We also support both `WHERE` and `GROUPBY` clauses. Because our focus is on the join itself, we ignore nested queries and only consider flat (or flattened) queries. We primarily focus on two-way joins. However, our results extend to multi-way joins with the same join column(s).

Aggregate Functions— Most AQP systems do not support extreme statistics, such as `Min` or `Max` [45]. Likewise, we only consider non-extreme aggregates, and primarily focus on the three basic functions, `COUNT`, `SUM`, and `AVG`. However, we expect our techniques to easily extend to other mean-like statistics as well, such as `VAR`, `STDEV`, and `PERCENTILE`.

3. HARDNESS

In this section, we explain why providing a large output size is insufficient for approximating joins, and formally show the hardness of approximating common aggregates based on the theory of communication complexity.

3.1 Output Size

Uniform sampling leads to small output size. If we sample at a rate q from both table T_1 and table T_2, the join of samples contains only q^2 fraction of $T_1 \bowtie T_2$ in expectation. Moreover, the join of two independent samples of the original tables is in general not an independent sample of $T_1 \bowtie T_2$, which hurts the sample quality. In contrast, universe sampling [31, 37] with sample rate p can, in expectation, sample a p fraction of $T_1 \bowtie T_2$. We prove that this is optimal (all omitted proofs are deferred to our report [35]).

Theorem 1. *No sampling scheme with sample rate α can guarantee more than α fraction of $T_1 \bowtie T_2$ in expectation for all possible inputs.*

However, a large number of tuples retained in the join does not imply that the original join query can be accurately approximated. As pointed out in [18], universe sampling shows poor performance in approximating queries when the frequencies of keys are concentrated on a few elements. Consider the following extreme example with tables T_1 and T_2, each comprised of n tuples with a single value 1 in their join key. In this example, universe sampling with the sampling rate p produces an estimator of variance n^4/p, while uniform sampling with rate q has a variance of n^2/q^2, which is much lower when $p = q$ and n is large. Thus, a larger output size does not necessarily lead to a better approximation of the query.

3.2 Approximating Aggregate Queries

In this section, we focus on the core question: why is approximating common aggregates (e.g., `COUNT`, `SUM` and `AVG`) hard when using a small sample (or more generally, a small summary)? We address this question using the theory of communication complexity. Specifically, to show that computing `COUNT` on a join is hard, we reduce it to set intersection, a canonically hard problem in communication complexity. Assume that both Alice and Bob each hold a set of size k, say A and B, respectively. They aim to estimate the size of $t = |A \cap B|$. Pagh et. al [47] show that if Alice only sends a small summary to Bob, any unbiased estimator that Bob uses will have a large variance.

Theorem 2 (See [47])**.** *Any one-way communication protocol that estimates t within relative error δ with probability at least $2/3$ must send at least $\Omega(k/(t\delta^2))n$ bits.*

Corollary 3. *Any estimator to $|A \cap B|$ produced by Bob that is based on an s-bits summary by Alice must have a variance of at least $\Omega(kt/s)$.*

Any sample of size s can be encoded using $O(\log \binom{k}{s})$ bits, implying that any estimator to `COUNT` that is based on a sample of size s from one of the tables must have a variance of at least $\Omega(kt/s)$.

Estimating `SUM` queries is at least as hard as estimating `COUNT` queries, since any `COUNT` can be reduced to a `SUM` by setting all entries in the `SUM` column to 1.

From the hard instance of set intersection, we can also derive a hard instance for `AVG` queries. Based on Theorem 2, any summary of T_1 that can distinguish between intersection size $t(1+\delta)$ and $t(1-\delta)$ must be at least of size $\Omega(k/(t\delta^2))$ bits. Now we reduce this problem to estimating an `AVG` query.

Here, the two tables consist of $k + \sqrt{t}$ tuples each. The first k tuples of T_1 and T_2 are from the hard instance of set intersection, and the values of their `AVG` column are set to $2r$. The join column of the last $\sqrt{t}$ tuples is set to some common key v' that is in the first k tuples, and their `AVG` column is set to 0. Therefore, the intersection size from the first k tuples is at least $t(1+\delta)$ (or at most $(t(1-\delta))$) if and only if the result of the `AVG` query is at least $\frac{2rt(1+\delta)}{t(2+\delta)} = (1+O(\delta))r$ (or at most $\frac{2rt(1+\delta)}{t(2+\delta)} = (1-O(\delta)r)$). By re-scaling δ by a constant factor, we can get the following theorem:

Theorem 4. *Any summary of T_1 that can estimate an `AVG` query with precision δ with probability at least $2/3$ must have a size of at least $\Omega(n/(t\delta^2))$.*

4. GENERIC SAMPLING SCHEME

To formally argue about the optimality of a sampling strategy, we must first define a *class* of sampling schemes. As discussed in Section 2.1, there are three well-known sampling operators: stratified, universe, and Bernoulli (uniform). However, these atomic operators can themselves be combined. For example, one can apply universe sampling of rate 0.1 and then Bernoulli sampling of rate 0.2 for an overall effective sampling rate of 0.02.[3] To account for such hybrid schemes, we define a generic scheme that combines universe and Bernoulli sampling, called UBS.[4] We also define a more generic scheme that combines all three of stratified, universe and Bernoulli sampling, called SUBS. It is easy to show that the basic sampling operators are a special case of SUBS. First, we define the effective sample rate.

Definition 5 (Effective sampling rate)**.** *We define the effective sampling rate of a sampling scheme as the expected ratio of the size of the resulting sample to that of the original table.*

Definition 6 (Universe-Bernoulli Sampling (UBS))**.** *Given a table T and a column (or set of columns) J in T, a UBS scheme is defined by a pair (p, q), where $0 < p \leq 1$ is a universe sampling rate and $0 < q \leq 1$ is a Bernoulli (or uniform) sampling rate. Let $h : \mathcal{U} \mapsto [0, 1]$ be a perfect hash function. Then, a sample of T produced by this scheme, $S = UBS_{p,q}(T, J)$, is produced as follows:*

Algorithm 1 $\mathrm{UBS}_{p,q}(T, J)$

```
S ← ∅  for each tuple t do
    if h(t.J) < p then
        Include t in S independently w/ prob. q.
    end
end
```

It is easy to see that the effective sampling rate of a UBS scheme (p, q) is $p \cdot q$. Thus, the effective sampling rate here is independent of the actual distribution of the values in the table (and column(s) J).

The goal of this sampling paradigm is to optimize the trade-off between universe sampling and Bernoulli sampling in different instances. At one extreme, when each join value appears exactly once in both table, universe sampling leads to lower variance than Bernoulli sampling. This is because independent Bernoulli sampling has trouble matching tuples with the same join value, while universe sampling guarantees that when a tuple is sampled, all matching tuples in the other table are also sampled. At the other extreme, if all tuples have the same join value in both tables (i.e., the join becomes a Cartesian product of the two tables), universe sampling will either sample the entire join, or sample nothing at all, while uniform sampling will have a sample size concentrated around qN, thus giving an estimator of much lower variance. In section 5.1 to 5.3, we give a comprehensive discussion on how to optimize p and q for different tables and different queries.

The Stratified-Universe-Bernoulli Sampling Scheme applies to a table T that is divided into K groups (i.e., strata), denoted as $G_1, G_2, \ldots, G_k$.

[3]Statistically, it does not matter which sampling is applied first: whether a tuple passes the universe sampler and whether it passes the Bernoulli sampler are completely independent decisions, and hence, the output distribution is the same. Here, we apply universe sampling first only for convenience and without loss of generality.

[4]Even if we do not care about output cardinality, universe sampling can still help improve the approximation quality. For example, given two tables of size n with a one-to-one join relationship, the count estimator's variance is n/q^2 under Bernoulli sampling but n/p under universe sampling, which is much lower when p=q.

Definition 7 (Stratified-Universe-Bernoulli Sampling (SUBS)). *Given a table T of N rows and a column (or set of columns) J in T, a SUBS scheme is defined by a tuple $(p_1, p_2, \ldots, p_K, q_1, q_2, \ldots, q_K)$, where $0<p_i, q_i \leq 1$ are the universe and Bernoulli sampling rates. Given a perfect hash function h: $\mathcal{U} \mapsto [0, 1]$, a sample of T produced by this scheme, $S = UBS_{p,q}(T, J)$, is produced as follows:*

Algorithm 2 $\text{SUBS}_{p_1,\ldots,p_K,q_1,\ldots,q_K}(T, G, J)$

```
S ← ∅
for each group G_i do
    for each tuple t in G_i do
        if h(t.J) < p_i then
            Include t in S independently w/ prob. q_i.
        end
    end
end
```

Let $|G_i|$ denote the number of tuples in group G_i. Then the effective sampling rate of a SUBS scheme is $\sum_i p_i \cdot q_i \cdot |G_i|/N$. We call $\epsilon_i = p \cdot q_i$ the effective sampling rate for group G_i.

In both UBS and SUBS schemes, the user specifies ϵ as their desired sampling budget, given which our goal is to determine optimal sampling parameters p and q (or p_i and q_i values) such that the variance of our join estimator is minimized. In Section 5, we derive the optimal p and q for UBS. For SUBS, in addition to ϵ, the user also provides two additional parameters k_{key} and k_{tuple} (explained below). Next, we show how to determine the effective sampling rate ϵ_i for each group G_i based on these parameters in SUBS. Given ϵ_i for each group, the problem is then reduced to finding the optimal parameters for UBS for that group (i.e., p_i and q_i). Moreover, as we will show in Sections 5.1–5.3, particularly in Lemma 9, the universe sampling rate for every group must be the same, and must be the same as the universe sampling rate of the other table in two-way joins. Hence, we use a single universe sampling rate $p = p_1 = \ldots = p_k$ across all groups.

As mentioned in Section 2.1, k_{tuple} is a user-specified lower bound on the minimum number of tuples[5] in each group the sample must retain. k_{key} is an additional user-specified parameter required for the SUBS scheme. It specifies a threshold at which to activate the universe sampler. In particular, if a group contains too few (i.e., less than k_{key}) *join keys*, we do not perform any universe sampling as it will have a high chance of filtering out all tuples. Hence, we apply universe sampling only to those groups with $\geq k_{\text{key}}$ join keys. For groups with fewer than k_{key} join keys, we will only apply Bernoulli sampling with rate ϵ_i.

We call a group *large* if it contains at least k_{key} join keys, otherwise, we call it a *small* group. We use N_b to denote the total number of tuples in all large groups, and N_s to denote the total number of tuples in all small groups. Similarly, let M_b and M_s denote the number of large and small groups, respectively. Then, we decide the sampling budget ϵ_i for each group G_i as follows:

1. If $M_s k_{\text{tuple}} > \epsilon N_s$ or $M_b k_{\text{tuple}} > \epsilon N_b$, we notify the user that creating a sample given their parameters is infeasible.
2. Otherwise,
   - Let $\epsilon'_s = \frac{K_s \cdot k_{\text{tuple}}}{N_s}$ and let $\epsilon''_s = \epsilon - \epsilon'_s$. Then for each small group G_i, the sampling budget is $\epsilon_i = \frac{k_{\text{tuple}}}{|G_i|} + \epsilon''_s$.
   - Let $\epsilon'_b = \frac{K_b \cdot k_{\text{tuple}}}{N_b}$ and let $\epsilon''_b = \epsilon - \epsilon'_b$. Then for each large group G_i, the sampling budget is $\epsilon_i = \frac{k_{\text{tuple}}}{|G_i|} + \epsilon''_b$.

Once ϵ_i is determined for each group, the problem of deciding optimal SUBS parameters is reduced to deciding the optimal SUBS parameters for K separate groups. This effective sampling rate ϵ_i guarantees that each large group will have at least t tuples in the sample on average, and the remaining budget is divided evenly. Thus, the corresponding uniform sampling rate for each large group is $q_i = \epsilon_i/p$. Moreover, we pose the constraint that the universe sampling rate p should be at least $1/s$ to guarantee that, on average, there is at least one join key passing through the universe sampler.

For small groups, we simply apply uniform sampling with rate ϵ_i. This is equivalent to setting $p = 1$ for these groups.

Overall, this strategy provides the following guarantees:

1. Each group will have at least t tuples in the sample, on average.
2. The probability of each group being missed is at most $(1 - 1/s)^s < 0.367$. In general, if we set $p > c/s$ for some constant $c > 1$, this probability will become 0.367^c.
3. The approximation of the original query will be optimal in terms of its variance (see Sections 5.1–5.3).

5. OPTIMAL SAMPLING

As shown in Section 4, finding the optimal sampling parameters within the SUBS scheme can be reduced to finding those within the UBS scheme. Thus, in this section, we focus on deriving the UBS parameters that minimize error for each aggregation type (`COUNT`, `SUM`, and `AVG`). Initially, we also assume there is no `WHERE` clause. Later, in Section 6, we show how to handle `WHERE` conditions and how to create a single sample instead of creating one per each aggregation type and `WHERE` condition.

Centralized vs. Decentralized— For each aggregation type, we analyze two scenarios: centralized and decentralized. Centralized setting is when the frequencies of the join keys in both tables are known. This represents situations where both tables are stored on the same server, or each server communicates its full frequency statistics to other parties. Decentralized setting is a scenario where the two tables are each stored on a separate server [61], and exchanging full frequency statistics across the network is costly.[6]

Decentralized Protocols— In a decentralized setting, each party (i.e., server) only has access to full statistics of its own table (e.g., frequencies, join column distribution). The goal then is for each party to determine its sampling strategy, while minimizing communications with the other party. Depending on the amount of information exchanged, one can pursue different protocols for achieving this goal. In this paper, we study a simple sampling protocol, which we call DICTATORSHIP. Here, one server, say `party1`, is chosen as the dictator. We also assume that the parties know each other's sampling budgets and table sizes (ϵ_1, ϵ_2, $|T_1|$, and $|T_2|$). The dictator observes the distributional information of its own table, say T_1, and decides a shared universe sampling rate p between $\max\{\epsilon_1, \epsilon_2\}$ and 1. This p is sent to the other server (`party2`) and both servers use p as their universe sampling rate.[7] Their uniform sampling rates will thus be $q_1 = \epsilon_1/p$ and $q_2 = \epsilon_2/p$, respectively.

Since `party1` only has T_1's frequency information, it chooses an optimal value of p that minimizes the *worst case* variance of

[5]The lower bound holds only on average, due to the probabilistic nature of sampling.

[6]Here, we focus on two servers, but the math can easily be generalized to decentralized networks of multiple servers.

[7]Using the same universe sampling rate is justified by Lemma 9.

$\hat{J}_{agg}$, i.e., the variance when the frequencies in T_2 are chosen adversarially. This can be formulated as a robust optimization [44]:

$$p^* = \underset{\max\{\epsilon_1,\epsilon_2\}\le p\le 1}{\arg\min} \max_b \mathrm{Var}[\hat{J}_{agg}] \quad (1)$$

where b ranges over all possible frequency vectors of T_2. In the rest of this paper, we use DICTATORSHIP in our decentralized analysis (we defer more complex protocols to [35]).

5.1 Join Size Estimation: Count on Joins

We start by considering the following simplified query:

```
select count(*) from T1 join T2 on J
```

where T_1 and T_2 are two tables joined on column(s) J. Consider $S_1 = \mathrm{UBS}_{(p_1,q_1)}(T_1, J)$ and $S_2 = \mathrm{UBS}_{(p_2,q_2)}(T_2, J)$. Then, we can define an unbiased estimator for the above query, $E_{\mathrm{count}} = |T_1 \bowtie_J T_2|$, using S_1 and S_2 as follows. Observe that given any pair of tuples $t_1 \in T_1$ and $t_2 \in T_2$, where $t_1.J = t_2.J$, the probability that (t_1, t_2) enters $S_1 \bowtie S_2$ is $p_{\min} q_1 q_2$, where $p_{\min} = \min\{p_1, p_2\}$. Hence, the following is an unbiased estimator for E_{count}.

$$\hat{J}_{\mathrm{count}}(p_1, q_1, p_2, q_2, S_1, S_2) = \frac{1}{p_{\min} q_1 q_2} |S_1 \bowtie S_2|. \quad (2)$$

When the arguments $p_1, q_1, p_2, q_2, S_1, S_2$ are clear from the context, we omit them and simply write $\hat{J}_{\mathrm{count}}$.

Lemma 8. *Let $S_1 = UBS_{p_1,q_1}(T_1, J)$ and $S_2 = UBS_{p_2,q_2}(T_2, J)$. The variance of $\hat{J}_{count}$ is as follows:*

$$\mathrm{Var}(\hat{J}_{count}) = \frac{1-p}{p}\gamma_{2,2} + \frac{1-q_2}{pq_2}\gamma_{2,1} + \frac{1-q_1}{pq_1}\gamma_{1,2} + \frac{(1-q_1)(1-q_2)}{pq_1q_2}\gamma_{1,1}.$$

where $\gamma_{i,j} = \sum_v a_v^i b_v^j$.

To minimize $\mathrm{Var}(\hat{J}_{\mathrm{count}})$ under a fixed sampling budget, the two tables should always use the same *universe* sampling rate. If $p_1 > p_2$, the *effective universe sampling rate* is only p_2, i.e., only p_2 fraction of the join keys inside T_1 appear in the join of the samples, and the remaining $p_1 - p_2$ fraction is simply *wasted*. Then, we can change the universe sampling rate of T_1 to p_2 and increase its uniform sampling rate to obtain a lower variance.

Lemma 9. *Given tables T_1, T_2 joined on column(s) J, a fixed sampling parameter (p_1, q_1) for T_1, and a fixed effective sampling rate ϵ_2 for T_2, the variance of $\hat{J}_{count}$ is minimized when T_2 uses p_1 as its universe sampling rate and correspondingly ϵ_2/p_1 as its uniform sampling rate.*

Note that Lemma 9 applies to both centralized and decentralized settings, i.e., it applies to any feasible sampling parameter (p_1, q_1) and (p_2, q_2), regardless of how the sampling parameter is decided. Next, we analyze each setting.

5.1.1 Centralized Sampling for Count

We have the following result.

Theorem 10. *When T_1 and T_2 use sampling parameters $(p, \epsilon_1/p)$ and $(p, \epsilon_2/p)$, $\hat{J}_{count}$'s variance is given by:*

$$\mathrm{Var}[\hat{J}_{count}] = (\frac{1}{p} - 1)\gamma_{2,2} + (\frac{1}{\epsilon_2} - \frac{1}{p})\gamma_{2,1} + (\frac{1}{\epsilon_1} - \frac{1}{p})\gamma_{1,2} + (\frac{p}{\epsilon_1\epsilon_2} - \frac{1}{\epsilon_1} - \frac{1}{\epsilon_2} + \frac{1}{p})\gamma_{1,1}.$$

Since each term in Theorem 10 that depends on p is proportional either to p or $1/p$, to find a p that minimizes the variance, one can simply set the first order derivatives (with respect to p) to 0.

Theorem 11. *Let T_1 and T_2 be two tables joined on column(s) J. Let a_v and b_v be the frequency of value v in column(s) J of tables T_1 and T_2, respectively. Given their sampling rates ϵ_1 and ϵ_2, the optimal sampling parameters (p_1, q_1) and (p_2, q_2) are given by:*

$$p_1 = p_2 = \min\{1, \max\{\epsilon_1, \epsilon_2, \sqrt{\frac{\epsilon_1\epsilon_2\gamma_{2,2} - \gamma_{1,2} - \gamma_{2,1} + \gamma_{1,1}}{\gamma_{1,1}}}\}\}$$

and $q_1 = \epsilon_1/p$, $q_2 = \epsilon_2/p$.

Substituting this into Lemma 8, the resulting variance is only a constant factor of Theorem 2's theoretical limit. For instance, consider a primary-key-foreign-key join query where $a_v \in \{0, 1\}$ and b_v is smaller than some constant, say 5, and $\epsilon_1 = \epsilon_2 = \epsilon$ for any ϵ, Theorem 11 chooses $p_1 = p_2 = \epsilon$. Then the variance given by Theorem 10 becomes $(1/\epsilon - 1)J$ where $J = \sum_v a_v b_v$ is the size of the join. Since ϵ is the expected ratio of the sample to table size, the expression $(1/\epsilon - 1)J$ matches the lower bound in Corollary 3 except for a constant factor.

5.1.2 Decentralized Sampling for Count

Motivated by Lemma 9, the DICTATORSHIP protocol uses the same universe sampling rate p for both parties in the decentralized setting, by solving the following robust optimization problem:

$$\underset{\max\{\epsilon_1,\epsilon_2\}\le p\le 1}{\arg\min} \max_b \mathrm{Var}[\hat{J}_{\mathrm{count}}]$$

Based on Lemma 8 and 11, given the effective sampling rates ϵ_1 and ϵ_2, we can express $\mathrm{Var}[\hat{J}_{\mathrm{count}}]$ as a function of frequencies $\{a_v\}$ and $\{b_v\}$, and universe sampling rate p as follows.

$$\mathrm{Var}[\hat{J}_{\mathrm{count}}] = (\frac{1}{p} - 1)\gamma_{2,2} + (\frac{1}{\epsilon_2} - \frac{1}{p})\gamma_{2,1} + (\frac{1}{\epsilon_1} - \frac{1}{p})\gamma_{1,2} + (\frac{p}{\epsilon_1\epsilon_2} - \frac{1}{\epsilon_1} - \frac{1}{\epsilon_2} + \frac{1}{p})\gamma_{1,1}. \quad (3)$$

Lemma 12. *Let a_* be the maximum frequency in table T_1, v_* be any value that has that frequency, and n_b be the total number of tuples in T_2. The optimal value for $\max_{\boldsymbol{b}\in\mathcal{K}_{n_b}} \mathrm{Var}[\hat{J}_{count}]$ is given by $(\frac{1}{p} - 1)a_*^2 n_b^2 + (\frac{1}{\epsilon_2} - \frac{1}{p})a_*^2 n_b + (\frac{1}{\epsilon_1} - \frac{1}{p})a_* n_b^2 + (\frac{p}{\epsilon_1\epsilon_2} - \frac{1}{\epsilon_1} - \frac{1}{\epsilon_2} - \frac{1}{p})a_* n_b$*

In equation (3), given $\{a_v\}$ and a fixed p, the variance is a convex function of the frequency vector $\{b_v\}$. Thus, the frequency vector $\{b_v\}$ that maximizes the variance, i.e., the worst case $\{b_v\}$, is one where exactly one join key has a non zero frequency. This join key should be the one with the maximum frequency in T_1. This is not a representative case and using it to decide a sampling rate might drastically hinder the performance on average. We therefore require that both servers also share a simple piece of information regarding the *maximum frequency* of the join keys in each table, say $F_a = \max_v a_v$ and $F_b = \max_v b_v$. With this information, the new optimal sampling rate is given by:

Theorem 13. *Given ϵ_1 and ϵ_2, the optimal UBS parameter (p, q_1) and (p, q_2) for `COUNT` in the decentralized setting are given by*

$$p = \min\{1, \max\{\epsilon_1, \epsilon_2, \sqrt{\epsilon_1\epsilon_2(F_aF_b - F_a - F_b + 1)}\}\}$$

and $q_1 = \epsilon_1/p$, $q_2 = \epsilon_2/p$.

5.2 Sum on Joins

Let E_{sum} be the output of the following simplified query:

```
select sum(T1.W)
from T1 join T2 on J
```

Let F be the sum of column W in the joined samples $S_1 \bowtie S_2$. Then, the following is an unbiased estimator for E_{sum}:

$$\hat{J}_{\text{sum}} = \frac{1}{p_{\min} q_1 q_2} F \tag{4}$$

where $p_{\min} = \min\{p_1, p_2\}$.

Lemma 14. $\mathrm{E}[\hat{J}_{sum}] = E_{\text{sum}}$.

Let μ_v and σ_v^2 be respectively the mean and variance of attribute W of the tuples in S_1 that have the join value v. Further, recall that a_v is the number of tuples in T_1 with join value v. The following lemma gives the variance of $\hat{J}_{\text{sum}}$.

Lemma 15. *The variance of $\hat{J}_{sum}$ is given by:*

$$\begin{aligned}\mathrm{Var}[\hat{J}_{sum}] = &\frac{1-q_2}{pq_2}\beta_1 + \frac{1-q_1}{pq_1}\beta_2 \\ &+\frac{(1-q_1)(1-q_2)}{pq_1q_2}\beta_3 + \frac{1-p}{p}\beta_4\end{aligned} \tag{5}$$

where $\beta_1 = \sum_v a_v^2 \mu_v^2 b_v$, $\beta_2 = a_v(\mu_v^2 + \sigma_v^2) b_v^2$, $\beta_3 = a_v(\mu_v^2 + \sigma_v^2) b_v$ and $\beta_4 = a_v^2 \mu_v^2 b_v^2$.

Analogous to Lemma 9, we have the following result.

Lemma 16. *Given tables T_1, T_2 joined on column(s) J, fixed sampling parameters (p_1, q_1) for T_1, and a fixed effective sampling rate $\epsilon_2 \le p_1$ for T_2, the variance of $\hat{J}_{sum}$ is minimized when T_2 also uses p_1 as its universe sampling rate and correspondingly, ϵ_2/p_1 as its uniform sampling rate.*

5.2.1 Centralized Sampling for Sum

Based on Lemma 16, we use the same universe sampling rate $p \ge \epsilon_1, \epsilon_2$ for both tables, with their corresponding uniform sampling rates being $q_1 = \epsilon_1/p$ and $q_2 = \epsilon_2/p$. Then we can further simplify equation 5 into:

Theorem 17. *When T_1 and T_2 both use the universe sampling rate p and respectively use the uniform sampling rate $q_1 = \epsilon_1/p$ and $q_2 = \epsilon_2/p$, the variance of $\hat{J}_{sum}$ is given by:*

$$\begin{aligned}\mathrm{Var}[\hat{J}_{sum}] = &\sum_v (\frac{1}{\epsilon_2} - \frac{1}{p})\beta_1 + (\frac{1}{\epsilon_1} - \frac{1}{p})\beta_2 \\ &+ (\frac{p}{\epsilon_1\epsilon_2} - \frac{1}{\epsilon_1} - \frac{1}{\epsilon_2} + \frac{1}{p})\beta_3 + (\frac{1}{p} - 1)\beta_4.\end{aligned}$$

Theorem 18. *Given effective sampling rates ϵ_1, ϵ_2, the optimal sampling parameters for `SUM` in a centralized setting are given by $p = \min\{1, \max\{\epsilon_1, \epsilon_2, \sqrt{\epsilon_1\epsilon_2 \frac{\beta_1+\beta_3-\beta_2-\beta_4}{\beta_3}}\}\}$, $q_1 = \frac{\epsilon_1}{p}$ and $q_2 = \frac{\epsilon_2}{p}$.*

5.2.2 Decentralized Sampling for Sum

Lemma 16 implies that, in a decentralized setting for `SUM` estimation, the universe sampling rate p must be decided by the party that has T_1, i.e., the table with the aggregate column.

Given a fixed T_1 and p, $\mathrm{Var}[\hat{J}_{\text{sum}}]$ is a strictly convex function of T_2's frequency vector. Hence, the worst case instance is a point distribution where all tuples in T_2 share the same join key. However, for `SUM`, the worst case distributions in T_2 are *not* the same for all possible sampling parameters p. Define $h_v(p)$ to be $\mathrm{Var}[\hat{J}_{\text{sum}}]$ as a function of p where T_2's frequency vector is all concentrated on the join key v, and define $h^*(p) = \max_v h_v(p)$. Since $h^*(p)$ is convex and piece-wise quadratic, its minimum can be attained using a sweepline algorithm (see [21, §8] for details). However, the memory usage is too costly in practice.

Therefore, we propose a simple sampling scheme whose worst case variance is at most twice the variance of the optimal scheme. Instead of using $h^*(p)$ to keep track of the maximum of all $h_v(p)$, we use an approximate $h'(p) = \max\{h_{v_1}(p), h_{v_2}(p)\}$, where $v_1 = \arg\max_v a_v^2\mu^2$ and $v_2 = \arg\max_v a_v(\mu_v^2 + \sigma_v^2)$ to approximate $h^*(p)$. The function h' is much simpler and its minimum can be easily found using quadratic equations and basic case analysis. For more details on the algorithm, refer to Appendix B in [35].

Let $p' = \arg\min h'(p)$ and $p^* = \arg\min h^*(p)$. We have:

Lemma 19. *For any $p \ge \epsilon_1, \epsilon_2$, we have $\frac{h^*(p)}{2} \le h'(p) \le h^*(p)$.*

Corollary 20. *We have: $h^*(p') \le 2h^*(p^*)$.*

5.3 Average on Joins

Let E_{avg} be the output of the following simplified query:

```
select avg(T1.W)
from T1 join T2 on J
```

In general, producing an unbiased estimator for `AVG` is hard.[8] Instead, we define and analyze the following estimator. Let S and C be the `SUM` and `COUNT` of column W in $S_1 \bowtie S_2$. We define our estimator as $\hat{J}_{\text{avg}} = S/C$. There are two advantages over using separate samples to evaluate `SUM` and `COUNT`: (1) we can use a larger sample to estimate both queries, and (2) since `SUM` and `COUNT` will be positively correlated, the variance of their ratio will be lower. Due to the lack of a close form expression for the variance of the ratio of two random variables, next we present a first order bivariate Taylor expansion to approximate the ratio.

Theorem 21. *Let S and C be random variables denoting the sum and cardinality of the join of two samples produced by applying UBS sampling parameters (p_1, q_1) to T_1 and (p_2, q_2) to T_2. Let $p_{\min} = \min\{p_1, p_2\}$. We have:*

$$\mathrm{Var}[S/C] = (\frac{E[S]^2}{E[C]^2})(\frac{\mathrm{Var}[S]}{E[S]^2} - \frac{2\mathrm{Cov}[S,C]}{E[S]E[C]} + \frac{\mathrm{Var}[C]}{\mathrm{E}[C]^2}) + R \tag{6}$$

where R is a remainder of lower order terms, and

$$\begin{aligned}\mathrm{Cov}[S,C] = &p_{\min} q_1 q_2 [(1-q_2)q_1 \sum_v a_v^2 \mu_v b_v + (1-q_1)q_2 \sum_v a_v \mu_v b_v^2 \\ &+(1-q_1)(1-q_2)\sum_v a_v \mu_v b_v + (1-p_{\min})q_1 q_2 \sum_v a_v^2 \mu_v b_v^2]\end{aligned}$$

and other expectation and variance terms are given by Theorems 10 and 17.

5.3.1 Centralized Sampling for Average

In a centralized setting where a_v, b_v, μ_v and σ_v are given for all v, every term in the expression $\frac{E[S]^2}{E[C]^2}(\frac{\mathrm{Var}[S]}{E[S]^2} - 2\frac{\mathrm{Cov}[S,C]}{E[S]E[C]} + \frac{\mathrm{Var}[C]}{\mathrm{E}[C]^2}$ that depends on p is proportional to either p or $1/p$.[9]

Thus, similar to Theorems 11 and 18, we can again find a p that minimizes the variance given by Theorem 21. We defer the exact expression of the optimal parameter to [35] due to space constraints.

[8] The denominator, i.e., the size of the sampled join, can even be zero. Furthermore, the expectation of a random variable's reciprocal is not equal to the reciprocal of its expectation.

[9] Notice that $E[S]/E[C]$ is independent of p.

5.3.2 Decentralized Sampling for Average

Minimizing the *worst case* variance for AVG (for the decentralized setting) is much more involved than the average case. In most cases, the objective function (variance) is neither convex nor concave in T_2's frequencies. However, note that every term in Theorem 21 is an inner product $\langle x, y \rangle$, where x and y are two vectors stored on party1 and party2, respectively. Fortunately, inner products can be approximated by transferring a very small amount of information using the AMS sketch [11, 24]. With such a sketch, we can derive an approximate sampling rate without communicating the full frequency statistics.

6. MULTIPLE QUERIES AND FILTERS

Creating a separate sample for each combination of aggregation function, aggregation column, and WHERE clause is clearly impractical. In this section, we show how to create a single sample per join pattern that supports multiple queries at the cost of some possible loss of approximation quality. First, we ignore the WHERE clauses and then show how they can be handled too.

Multiple Tables and Queries— We formulate our input as a graph G=$\langle V, E \rangle$. The vertex set V is the set of all table and join key pairs, and the edge set E corresponds to all join queries of interest. Specifically, for every join query between tables T_1 and T_2 on $J_1 = J_2$, we have a corresponding edge e between vertices $(T_1, J_1) \in V$ and $(T_2, J_2) \in V$ (henceforth, we will use a query and its corresponding edge interchangeably). This means G is a multigraph, with potentially parallel edges or self-loops. For each vertex $v = (T, J) \in V$, we must output a sampling budget ϵ_v as well as the corresponding universe sampling rate p_v, which will be used to create a sample $S = \text{UBS}_{p_v, \epsilon_v/p_v}(T, J)$. This sample will be used for any query that involves a join with T on column(s) J.

According to Lemmas 8 and 15, and Theorem 21, for each edge e=$(v_1,v_2)$$\in E$, we can express the estimator variance of its corresponding query as a function of $\epsilon_{v_1}, \epsilon_{v_2}, p_{v_1}, p_{v_2}$ and p_e, where p_e is an auxiliary variable denoting the minimum of p_1 and p_2:

$$f_e(p, \epsilon_{v_1}, \epsilon_{v_2}, p_{v_1}, p_{v_2}) = \frac{1}{p_e}\left(A_e + B_e \frac{p_1}{\epsilon_{v_1}} + C_e \frac{p_2}{\epsilon_{v_2}} + D_e \frac{p_1 p_2}{\epsilon_{v_1}\epsilon_{v_2}}\right) \tag{7}$$

where A_e, B_e, C_e, D_e are constants that depend on the distributional information of the tables in v_1 and v_2. To cast this as an optimization problem, we also take in a user specified weight ω_e for each edge e and express our objective as:

$$F = \sum_{e=(v_1,v_2)\in E} \omega_e f_e(p_e, \epsilon_{v_1}, \epsilon_{v_2}, p_{v_1}, p_{v_2}) \tag{8}$$

The choice of ω_e values is up to the user. For example, they can all be set to 1, or to the relative frequency, importance, or probability of appearance (e.g., based on past workloads) of the query corresponding to e. Then, to find the optimal sampling parameters we solve the following optimization:

$$\min_{\epsilon_v, p_v, p_e} \quad F \qquad \text{subject to} \sum_{v=(T,J)\in V} \epsilon_v \cdot \text{size}(T) \le B \tag{9}$$

where $size(T)$ is the storage footprint of table T, and B is the overall storage budget for creating samples. Note that by replacing the non-linear $p_e = \min(p_{v_1}, p_{v_1})$ constraints with $p_e \le p_{v_1}$ and $p_e \le p_{v_2}$, (9) is reduced to a smooth optimization problem, which can be solved numerically with off-the-shelf solvers [14].

Known Filters— To incorporate WHERE clauses, we simply regard a query with a filter c on $T_1 \bowtie T_2$ as a query without a filter but on a sub-table that satisfies c, namely $T' = \sigma_c(T_1 \bowtie T_2)$.

Unknown Filters with Distributional Information— When the columns appearing in the WHERE clause can be predicted but the exact constants are unknown, a similar technique can be applied. For example, if an equality constraint $C > x$ is anticipated but x may take on 100 different values, we can *conceptually* treat it as 100 separate queries, each with a different value of x in its WHERE clause. This reduces our problem to that of sampling for multiple queries without a WHERE clause, which we know how to handle using equation (8).[10] Here, the weight ω_i can be used to exploit any distributional information that might be available. In general, ω_i should be set to reflect the probability of each possible WHERE clause appearing in the future. For example, if there are R possible WHERE clauses and all are equally likely, we can set $\omega_i = 1/R$, but if popular values in a column are more likely to appear in the filters, we can use the column's histogram to assign ω_i.

Unknown Filters— When there is no information regarding the columns (or their values) in future filters, we can take a different approach. Since the estimator variance is a monotone function in the frequencies of each join key (see Theorem 10, Theorems 17 and 21), the larger the frequencies, the larger the variance. This means the worst case variance always happens when the WHERE clause selects all tuples from the original table. Hence, in the absence of any distributional information regarding future WHERE clauses, we can simply focus on the original query without any filters to minimize our worst case variance.

7. EXPERIMENTS

Our experiments aim to answer the following questions:

(i) How does our optimal sampling compare to other baselines in centralized and decentralized settings? (§7.2, §7.3)

(ii) How well does our optimal UBS sampling handle join queries with filters? (§7.4)

(iii) How does our optimal UBS sampling perform when using a single sample for multiple queries? (§7.5)

(iv) How does our optimal SUBS sampling compare to existing stratified sampling strategies? (§7.6)

(v) How much does a decentralized setting reduce the resource consumption and sample creation overhead? (§7.7)

7.1 Experiment Setup

Hardware and Software— We borrowed a cluster of 18 *c220g5* nodes from CloudLab [4]. Each node was equipped with an Intel Xeon Silver 4114 processor with 10 cores (2.2Ghz each) and 192GB of RAM. We used Impala 2.12.0 as our backend database to store data and execute queries.

Datasets— We used several real-life and synthetic datasets:

1. **Instacart [1].** This is a real-world dataset from an online grocery. We used their *orders* and *order_products* tables (3M and 32M tuples, resp.), joined on *order_id*.
2. **Movielens [32].** This is a real-world movie rating dataset. We used their *ratings* and *movies* tables (27M and 58K tuples, resp.), joined on *movieid*.
3. **TPC-H [3]**. We used a scale factor of 1000, and joined *l_orderkey* of the fact table (*lineitem*, 6B tuples) with *o_orderkey* of the largest dimension table (*orders*, 1.5B tuples).

[10] Note that, even though each query in this case is on a different table, they are all sub-tables of the same original table, and hence their sampling rate p is the same.

Table 2: **Six UBS baselines, each with different p and q.**

	B_1	B_2	B_3	B_4	B_5	B_6
p	0.001	0.0015	0.003	0.333	0.6667	1.000
q	1.000	0.6667	0.333	0.003	0.0015	0.001

Table 3: **Optimal sampling parameters (centralized setting).**

Dataset	COUNT		SUM		AVG	
	p	q	p	q	p	q
S{*uniform,uniform*}	0.010	0.1	0.004	0.264	0.001	1.000
S{*uniform,normal*}	0.012	0.083	0.005	0.220	0.001	1.000
S{*uniform,power1*}	1.000	0.001	1.000	0.001	0.692	0.001
S{*uniform,power2*}	1.000	0.001	1.000	0.001	0.001	1.000
S{*normal,uniform*}	0.012	0.083	0.009	0.111	0.001	1.000
S{*normal,normal*}	0.014	0.069	0.011	0.093	0.001	1.000
S{*normal,power1*}	1.000	0.001	1.000	0.001	0.001	1.000
S{*normal,power2*}	1.000	0.001	1.000	0.001	0.001	1.000
S{*power1,uniform*}	1.000	0.001	1.000	0.001	0.001	1.000
S{*power1,normal*}	1.000	0.001	1.000	0.001	0.001	1.000
S{*power1,power1*}	1.000	0.001	1.000	0.001	0.001	1.000
S{*power2,uniform*}	1.000	0.001	1.000	0.001	0.001	1.000
S{*power2,normal*}	1.000	0.001	1.000	0.001	0.001	1.000
S{*power2,power2*}	1.000	0.001	1.000	0.001	0.001	1.000
Instacart	0.01	1.00	0.01	1.00	0.01	1.00
Movielens	0.1	1.00	0.1	1.00	0.1	1.00
TPC-H	0.001	1.00	0.001	1.00	0.001	1.00

4. **Synthetic.** To better control the join key distribution, we also generated several synthetic datasets, where tables T_1 and T_2 each had 100M tuples and a join column J. T_1 had an additional column W for aggregation, drawn from a power law distribution with range [1, 1000] and α=3.5. We varied the distribution of the join key in each table to be one of uniform, normal, or power law, creating different datasets (listed in Table 3). The values of column J were integers randomly drawn from [1, 10M] according to the chosen distribution. Whenever joining with *power2* (see below), we used 100K join keys in both relations. For normal distribution, we used a truncated distribution with σ=1000/5. We used two different variants of power law distribution for J, one with α=1.5 and 10M join keys (referred to as *power1*), and one with α=2.0 and 100K join keys (referred to as *power2*). We denote each synthetic dataset according to its tables' distributions, S{*distribution of* T_1*,distribution of* T_2}, e.g., S{*uniform,uniform*}.

Baselines— We compared our optimal UBS parameters (referred to as OPT) against six baselines. The UBS parameters of these baselines, $B_1, \ldots, B_6$, are listed in Table 2. B_1 and B_6 are simply universe and uniform sampling, respectively. $B_2, \ldots, B_5$ represent different hybrid variants of these sampling schemes. Sampling budgets were $\epsilon_1 = \epsilon_2 = 0.001$, except for Instacart and Movielens where, due to their small number of tuples, we used 0.01 and 0.1, respectively.

Implementation— We implemented our optimal parameter calculations in Python application. Our sample generation logic read required information, such as table size and join key frequencies, from the database, and then constructed SQL statements to build appropriate samples in the target database. We used Python to compute approximate answers from sample-based queries.

Variance Calculations— We generated β=500 pairs of samples for each experiment, and re-ran the queries on each pair, to calculate the variance of our approximations.

7.2 Join Approximation: Centralized Setting

Table 3 shows the sampling rates used by OPT for each dataset and aggregate function in the centralized setting. For Synthetic, the optimal parameters were some mixture of uniform and universe sampling when both tables were only moderately skewed (i.e., uniform or normal distributions) for COUNT and SUM, whereas it reduced to a simple uniform sampling for power law distribution. This is due to the higher probability of missing extremely popular join keys with universe sampling. To the contrary, for AVG, OPT reduced to a simple universe sampling in most cases. This is because maximizing the output size in this case was the best way to reduce variance. For the other datasets (Instacart, Movielens, and TPC-H), the optimal parameters led to universe sampling, regardless of aggregate type, and their joins were PK-FK, hence making uniform sampling less useful for the table with primary keys.

Figure 2 shows OPT's improvement over the baselines in terms of variance for COUNT queries. Each bar is also annotated with the relative percentage error of the corresponding baseline. OPT outperformed all baselines in most cases, achieving over 10x lower variance than the worst baseline. Figures 3 and 4 show the same experiment for SUM and AVG. In both cases, OPT achieved the minimum variance across all sampling strategies, except for AVG when T_1 or T_2 was a power law distribution. This is because OPT for AVG was calculated using a Taylor approximation, which is accurate only when the estimators of SUM and COUNT are both within the proximity of their true values. Moreover, sample variance converges slowly to the theoretical variance, particularly for skew distributions, such as power law. This is why estimated variances for OPT were not optimal for some Synthetic datasets. However, OPT still achieved the lowest variance across all real-world datasets, as shown in Figure 5. Here, for the selected join key, OPT determined that a full universe sampling was the best sampling scheme.

In summary, this experiment highlights OPT's ability in outperforming simple uniform or universe sampling—or choosing one of them, when optimal—for aggregates on joins.

7.3 Join Approximation: Decentralized

We evaluated both OPT and other baselines under a decentralized setting using Instacart and Synthetic datasets. Here, we constructed a possible worst case distribution for T_2 that was still somewhat realistic, given the distribution of T_1 and minimal information about T_2 (i..e, T_2's cardinality). To do this, we used the following steps: 1) let $J_{MAX(T_1)}$ be the most frequent join key value in T_1; 2) assign 75% of the join key values of T_2 to have the value of $J_{MAX(T_1)}$ and draw the rest of the join key values from a uniform distribution.

Figure 6 shows the results. For Synthetic, the OPT was the same under both settings whenever there was a power law distribution or the aggregate was AVG. This is because our assumption of the worst case distribution for T_2 was close to a power law distribution. For COUNT and SUM with Synthetic dataset, OPT in the decentralized setting had a much higher variance than OPT in the centralized setting when there was no power law distribution. With Instacart, OPT in the decentralized setting was the same as OPT in the centralized setting, which had the minimum variance among the baselines. This illustrates that OPT in the decentralized setting can perform well with real-world data where the joins are mostly PK-FK. This also shows that if a reasonable assumption is possible on the distribution of T_2, OPT can be as effective in the decentralized setting as it is in a centralized one, while requiring significantly less communication.

7.4 Join Approximation with Filters

To study OPT's effectiveness in the presence of filters, we used S{*uniform,uniform*} and Instacart datasets with ϵ=0.01. We added an extra column C to T_1 in S{*uniform,uniform*}, with in-

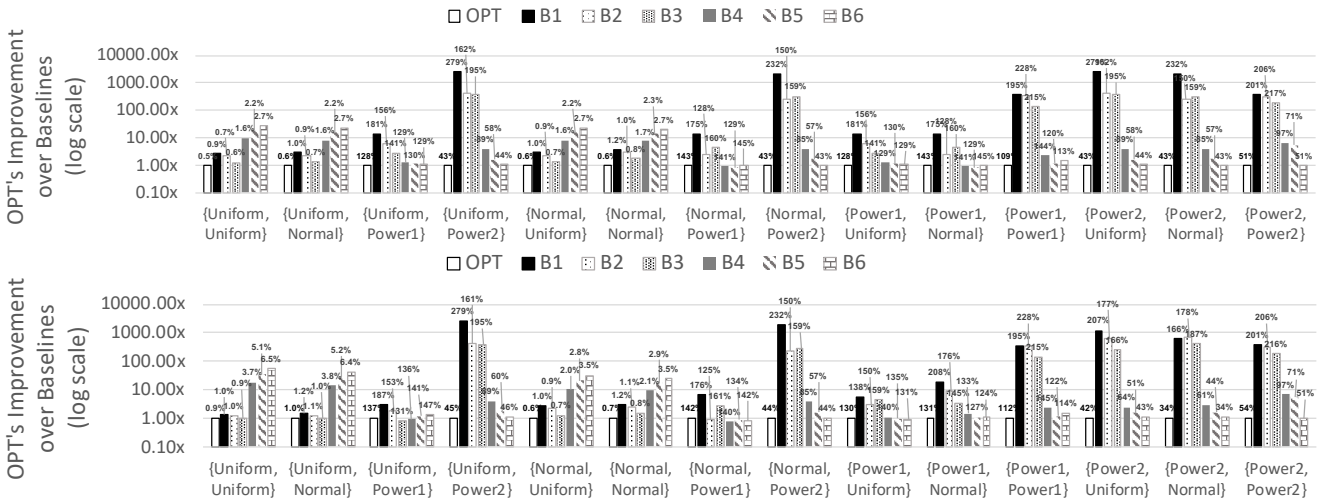


Figure 3: **OPT's improvement in terms of variance for SUM over six baselines with synthetic dataset (percentages are relative error).**

Table 4: **Optimal sampling parameters for $\mathcal{S}\{$*uniform,uniform*$\}$ for different distributions of the filtered column C.**

Dist. of C	COUNT		SUM		AVG	
	p	q	p	q	p	q
Uniform	0.010	1.000	0.010	1.000	0.010	1.000
Normal	0.018	0.555	0.015	0.648	0.010	1.000
Power law	0.051	0.195	0.050	0.201	0.010	1.000

Table 5: **Sampling parameters (p and q) of OPT using individual samples for different aggregates versus a combined sample ($\mathcal{S}\{$*normal,normal*$\}$ dataset).**

Scheme	COUNT		SUM		AVG	
	p	q	p	q	p	q
OPT (individual)	0.145	0.069	0.125	0.080	0.010	1.000
OPT (combined)	0.133	0.075	0.133	0.075	0.133	0.075

tegers in $[1, 100]$, and tried three distributions (uniform, normal, power law). For Instacart, we used the *order_hour_of_day* column for filtering, which had an almost normal distribution. We used an equality operator and chose the comparison value x uniformly at random. We calculated the average variance over all possible values of c.

Table 4 shows the sampling rates chosen by OPT, while Figure 7 shows OPT's improvement over baselines in terms of average variance. Again, OPT successfully achieved the lowest average variance among all baselines in all cases, up to 10x improvement compared to the worst baseline. This experiment confirms that UBS with OPT is highly effective for join approximation, even in the presence of filters.

7.5 Combining Samples

We evaluated the idea of using a single sample for multiple queries instead of generating individual samples for each query, as discussed in Section 6. Here, we use OPT (individual) and OPT (combined) to denote the use of one-sample-per-query and one-sample-for-multiple-queries, respectively. For OPT (combined), we considered a scenario where each of COUNT, SUM, and AVG is equally likely to appear. Table 5 reports the sampling rates chosen in each case. As shown in Figure 8, without having to generate an individual sample for each query, the variances of OPT (combined) were only slightly higher than those of OPT (individual). This experiment shows that it is possible to create a single sample for multiple queries without sacrificing too much optimality.

7.6 Stratified Sampling

We also evaluated SUBS for join queries with group-by. Here, we used the $\mathcal{S}\{$*normal,normal*$\}$ dataset, and added an extra group column G to T_1 with integers from 0 to 9 drawn from a power law distribution with $\alpha = 1.5$. This time we did not randomize the groups, i.e., G=0 had the most tuples and G=9 had the fewest. This was to study SUBS performance with respect to the different group sizes. As a baseline, we generated stratified samples for T_1 on G with $k_{\text{key}} = 100,000$ and uniform samples for T_2 with a 0.01 sampling budget. We denote this baseline as *SS_UF*. For SUBS, we used parameters that matched the sample size of *SS_UF*, i.e., $k_{\text{key}} = 100, k_{\text{tuple}} = 100,000$. Figure 9 shows the variance of query estimators for each of the 10 groups for different aggregations. As expected, SUBS with OPT achieved lower variances than *SS_UF* across all aggregates and groups with different sizes.

7.7 Overhead: Centralized vs. Decentralized

We compared the overhead of OPT in centralized versus decentralized settings, in terms of the sample creation time and resources, such as network and disk. OPT should have a much higher overhead in the centralized setting, as it requires full frequency information of every join key value in both tables. To quantify their overhead difference, we used Instacart and TPC-H, and created a pair of samples for SUM in each case. Here, the aggregation type did not matter, as the time spent calculating p and q was negligible compared to the time taken by transmitting the frequency vectors.

As shown in Figure 10, we measured the time for statistics acquisition, sampling rate calculation, and sample table creation. Here, the time taken by collecting the frequencies was the dominant factor. For Instacart, it took 65.16 secs from start to finish in the decentralized setting, compared to 99.98 secs in the centralized setting, showing 1.53x improvement in time. For TPC-H, it took 59.5 min in the decentralized setting, compared to 91.7 mins in the centralized, showing a speedup of 1.54x.

We also measured the total network and disk I/O usage across the entire cluster, as shown in Figure 11. For Instacart, compared to the decentralized setting, the centralized one used 3.66x (0.9 $\rightarrow$ 3.29 MB) more network and 2.22x (7.59 $\rightarrow$ 16.9 MB) more disk bandwidth. Overall, the overhead was less for TPC-H. The centralized in this case used 1.38x (243.39 $\rightarrow$ 337.04 MB) more network and 1.49x (519.03 $\rightarrow$ 776.58 MB) more disk bandwidth than the decentralized setting.

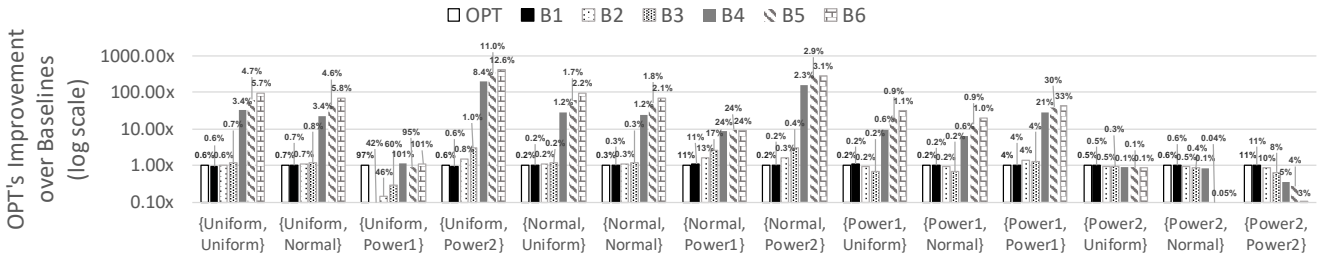
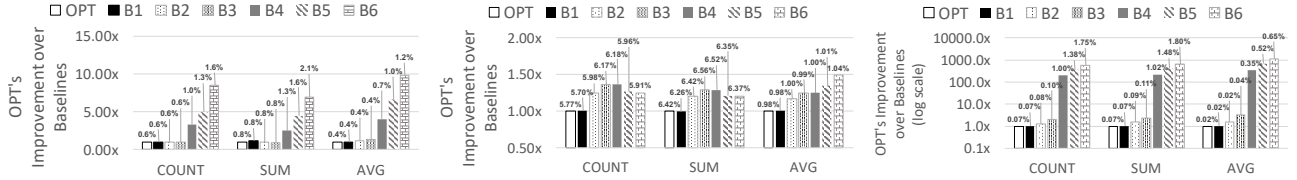
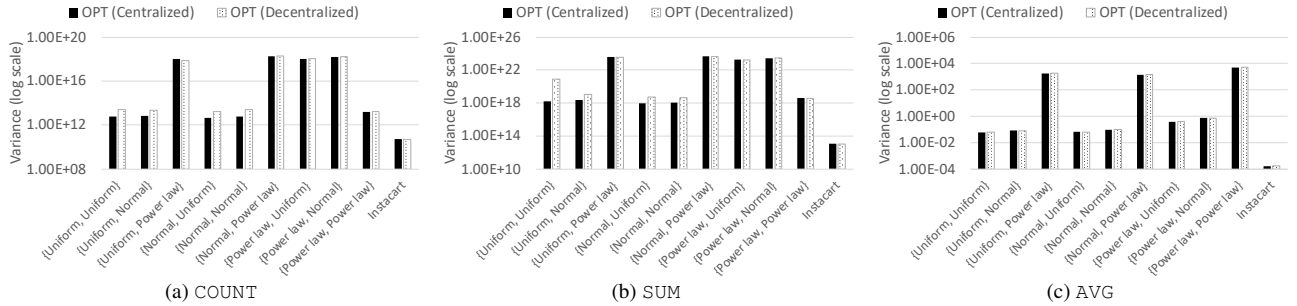


Figure 4: **OPT's improvement in terms of variance for AVG over six baselines with synthetic dataset (percentages are relative error).**



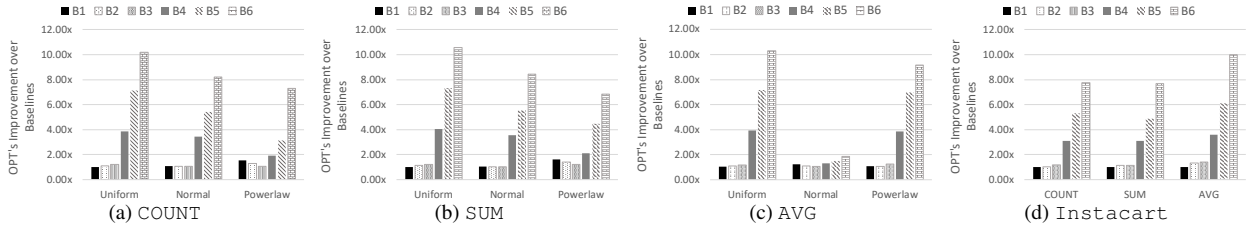
(a) Instacart (b) Movielens (c) TPC-H

Figure 5: **OPT's improvement in terms of variance over the baselines on benchmark datasets (percentages are relative error).**



(a) COUNT (b) SUM (c) AVG

Figure 6: **Variances of the query estimators for OPT in the centralized and decentralized settings.**



(a) COUNT (b) SUM (c) AVG (d) Instacart

Figure 7: **OPT's improvement in terms of the estimator's variance over six baselines in the presence of filters.**

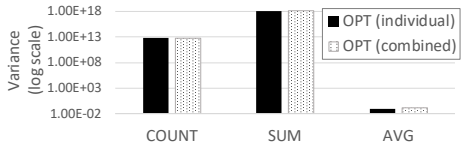


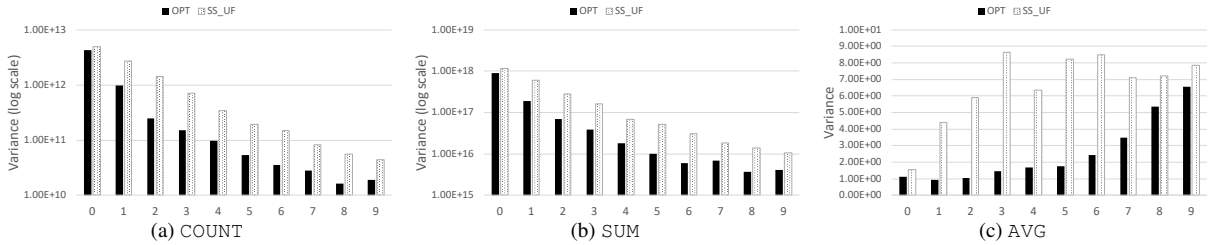
Figure 8: **Variance of the query estimators for OPT (individual) and OPT (combined) for the $S\{normal,normal\}$ dataset.**

This experiment shows the graceful tradeoff between the optimality of sampling and its overhead, making the decentralized variant an attractive choice for large datasets and distributed systems.

8. RELATED WORK

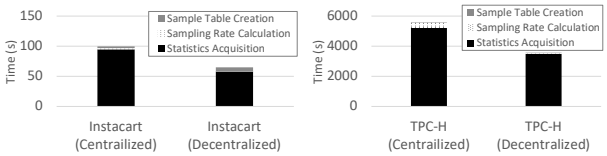
Online Sample-based Join Approximation— Ripple Join [29,42] is an online join algorithm that operates under the assumption that the tuples of the original tables are processed in a random order. Each time, it retrieves a random tuple (or a set of random tuples) from the tables, and then joins the new tuples with the previously read tuples and with each other. SMS [36] speeds up the hashed version of Ripple Join when hash tables exceed memory. Wander Join [40] tackles the problem of k-way chain join and eliminates the random order requirement of Ripple Join. However, it requires an index on every join column in each of the tables. Using indexes, Wander Join performs a set of random walks and obtains a *non-uniform* but independent sample of the join. Maintaining an approximation of the size of all partial joins can help overcome the non-uniformity problem [41, 63].

Offline Sample-based Join Approximation— AQUA [5] acknowledges the quadratic reduction and the non-uniformity of the output when joining two uniform random samples. The same authors propose *Join Synopsis* [6], which computes a sample of one of the tables and joins it with the other tables as a sample of the actual join. Chaudhuri et al. [17] also point out that a join of independent samples from two relations does not yield an independent sample of their join, and propose using precomputed statistics to overcome this problem. However, their solution can be quite costly, as it requires collecting full frequency information of the relation. Zhao et



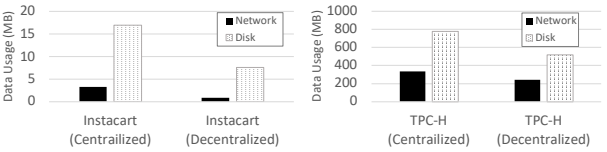
(a) COUNT (b) SUM (c) AVG

Figure 9: **Query estimator variance per group for for a group-by join aggregate using SUBS versus *SS_UF*.**



(a) Instacart (b) TPC-H

Figure 10: **Time taken to generate samples for `Instacart` and `TPC-H` in centralized vs. decentralized setting.**



(a) Instacart (b) TPC-H

Figure 11: **Total network and disk bandwidth used to generate samples for `Instacart` and `TPC-H`.**

al. [63] provide a better trade-off between sampling efficiency and the join size upper bound. Hashed sampling (a.k.a. universe) [31] is proposed in the context of selectivity estimation for set similarity queries. Block-level uniform sampling [16] is less accurate but more efficient than tuple-level sampling. Bi-level sampling [19,30] performs Bernoulli sampling at both the block- and tuple-level, as a trade-off between accuracy and I/O cost of sample generation.

AQP Systems on Join— Most AQP systems rely on sampling and support certain types of joins [5,8,15,27,37,41,50,54]. STRAT [15] discusses the use of uniform and stratified sampling, and how those can support certain types of join queries. More specifically, STRAT only supports PK-FK joins between a fact table and one or more dimension table(s). BlinkDB [8] extends STRAT and considers multiple stratified samples instead of a single one. As previously mentioned, AQUA [5] supports foreign key joins using join synopses. *Icicles* [27] samples tuples that are more likely to be required by future queries, but, similar to AQUA, only supports foreign key joins. PF-OLA [54] is a framework for parallel online aggregation. It studies parallel joins with group-bys, when partitions of the two tables fit in memory. XDB [41] integrates Wander Join in PostgreSQL. Quickr [37] does not create offline samples. Instead, it uses universe sampling to support equi-joins, where the group-by columns and the value of aggregates are not correlated with the join keys. VerdictDB [50] is a universal AQP framework that supports all three types of samples (uniform, universe, and stratified). VerdictDB utilizes a technique called *variational subsampling*, which creates subsamples of the sample such that it only requires a single join—instead of repeatedly joining the subsamples multiple times—to produce accurate aggregate approximations.

Join Cardinality Estimation— There is extensive work on join cardinality estimation (i.e.,`count(*)`) in the database community [10, 26, 38, 39, 53, 57, 58, 60] as an important step of the query optimization process for joins. Two-level sampling [18] first applies universe sampling to the join values, and then, for each join value sampled, it performs Bernoulli sampling. However, unlike our UBS scheme which applies the same rate to all keys, two-level sampling uses a different rate during its universe sampling for each join key. In other words, two-level sampling is a more complex scheme with significantly more parameters than UBS (which requires only two parameters, p and q), and is thus less amenable to efficient and decentralized implementation. Furthermore, two-level sampling applies two different sampling methods, whereas bi-level sampling [30] uses only Bernoulli sampling but at different granularity levels. End-biased sampling [26] samples each tuple with a probability proportional to the frequency of its join key. Index-based sampling [39] and deep learning [38] have also been utilized to improve cardinality estimates.

Theoretical Studies— The question about the limitation of sample-based approximation of joins, to the best of our knowledge, has not been asked in the theory community. However, the past work in communication complexity on set intersection and inner product estimation has implications for join approximation. In this problem, the Alice and Bob possess respectively two vectors x and y and they wish to compute their inner product $t = \langle x, y \rangle$ without exchanging the vector x and y. In the one-way model, Alice computes a summary $\beta(x)$ and sends it to Bob, who will estimate $\langle x, y \rangle$ using y and $\beta(x)$. For this problem, [47] shows that any estimator produced by s bits of communication has variance at least $\Omega(dt/s)$. Estimating inner product for $0, 1$ vectors is directly related to estimating `SUM` and `COUNT` for a PK-FK join. A natural question is whether the join is still hard even if frequencies are all larger than 1. Further, the question of whether estimating `AVG` is also hard is not answered by prior work.

9. CONCLUSION

Our goal in this paper was to improve our understanding of join approximation using offline samples, and formally address some of the key open questions faced by practitioners using and building AQP engines. We defined generic sampling schemes that cover the most common sampling strategies, as well as as their combinations. Within these schemes, we (1) provided an information-theoretical lower bound on the lowest error achievable by any offline sampling scheme, (2) derived optimal strategies that match this lower bound within a constant factor, and (3) offered a decentralized variant that requires minimal communication of statistics across the network. These results allow practitioners to quickly determine—e.g., based on the distribution of the join columns—if joining offline samples will be futile or will yield a reasonable accuracy. We also expect our hybrid samples to improve the accuracy of database learning [51] and selectivity estimation [52] for join queries.

10. ACKNOWLEDGEMENT

This research is in part supported by the National Science Foundation through grants 1553169 and 1629397.

11. REFERENCES

[1] The instacart online grocery shopping dataset 2017. https://www.instacart.com/datasets/grocery-shopping-2017. Accessed: 2019-07-20.

[2] Security on-demand announces acquisition of Infobright analytics & technology assets. https://tinyurl.com/y6ctn4vs.

[3] TPC-H Benchmark. http://www.tpc.org/tpch/.

[4] Cloudlab. https://www.cloudlab.us, 2019.

[5] S. Acharya, P. B. Gibbons, and V. Poosala. Aqua: A fast decision support system using approximate query answers. In *VLDB*, 1999.

[6] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approximate query answering. In *SIGMOD*, 1999.

[7] Sameer Agarwal, Henry Milner, Ariel Kleiner, Ameet Talwalkar, Michael Jordan, Samuel Madden, Barzan Mozafari, and Ion Stoica. Knowing when you're wrong: Building fast and reliable approximate query processing systems. In *SIGMOD*, 2014.

[8] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. BlinkDB: queries with bounded errors and bounded response times on very large data. In *EuroSys*, 2013.

[9] Sameer Agarwal, Aurojit Panda, Barzan Mozafari, Anand P. Iyer, Samuel Madden, and Ion Stoica. Blink and it's done: Interactive queries on very large data. *PVLDB*, 2012.

[10] Noga Alon, Phillip B Gibbons, Yossi Matias, and Mario Szegedy. Tracking join and self-join sizes in limited storage. *Journal of Computer and System Sciences*, 64, 2002.

[11] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58, 1999.

[12] Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM J. Comput.*, 42(4), 2013.

[13] Brian Babcock, Surajit Chaudhuri, and Gautam Das. Dynamic sample selection for approximate query processing. In *VLDB*, 2003.

[14] Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2014.

[15] Surajit Chaudhuri, Gautam Das, and Vivek Narasayya. Optimized stratified sampling for approximate query processing. *TODS*, 2007.

[16] Surajit Chaudhuri, Gautam Das, and Utkarsh Srivastava. Effective use of block-level sampling in statistics estimation. In *SIGMOD*, 2004.

[17] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. On random sampling over joins. In *SIGMOD*, 1999.

[18] Yu Chen and Ke Yi. Two-level sampling for join size estimation. In *SIGMOD*, 2017.

[19] Yu Cheng, Weijie Zhao, and Florin Rusu. Bi-level online aggregation on raw data. In *SSDBM*, 2017.

[20] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, and Russell Sears. Mapreduce online. In *NSDI*, 2010.

[21] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[22] Graham Cormode, Minos Garofalakis, Peter J Haas, and Chris Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4, 2012.

[23] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. Vizdom: Interactive analytics through pen and touch. *PVLDB*, 2015.

[24] Alin Dobra, Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Processing complex aggregate queries over data streams. In *SIGMOD*, 2002.

[25] Alin Dobra, Chris Jermaine, Florin Rusu, and Fei Xu. Turbo-charging estimate convergence in dbo. *PVLDB*, 2009.

[26] Cristian Estan and Jeffrey F. Naughton. End-biased samples for join cardinality estimation. In *ICDE*, 2006.

[27] Venkatesh Ganti, Mong-Li Lee, and Raghu Ramakrishnan. Icicles: Self-tuning samples for approximate query answering. In *VLDB*, 2000.

[28] Deepak Goyal. Approximate query processing at WalmartLabs. https://fifthelephant.talkfunnel.com/2018/43-approximate-query-processing.

[29] Peter J. Haas and Joseph M. Hellerstein. Ripple Joins for Online Aggregation. In *SIGMOD*, pages 287–298, 1999.

[30] Peter J Haas and Christian König. A bi-level bernoulli scheme for database sampling. In *SIGMOD*, 2004.

[31] Marios Hadjieleftheriou, Xiaohui Yu, Nick Koudas, and Divesh Srivastava. Hashed samples: selectivity estimators for set similarity selection queries. *PVLDB*, 2008.

[32] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *TIIS*, 2016.

[33] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. Online aggregation. In *SIGMOD*, 1997.

[34] Daniel G Horvitz and Donovan J Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American statistical Association*, 47, 1952.

[35] Dawei Huang, Dong Young Yoon, Seth Pettie, and Barzan Mozafari. Joins on samples: A theoretical guide for practitioners. https://arxiv.org/abs/1912.03443, 2019.

[36] Christopher Jermaine, Alin Dobra, Subramanian Arumugam, Shantanu Joshi, and Abhijit Pol. A disk-based join with probabilistic guarantees. In *SIGMOD*, 2005.

[37] Srikanth Kandula, Anil Shanbhag, Aleksandar Vitorovic, Matthaios Olma, Robert Grandl, Surajit Chaudhuri, and Bolin Ding. Quickr: Lazily approximating complex adhoc queries in bigdata clusters. In *SIGMOD*, 2016.

[38] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. Learned cardinalities: Estimating correlated joins with deep learning. *arXiv:1809.00677*, 2018.

[39] Viktor Leis, Bernhard Radke, Andrey Gubichev, Alfons Kemper, and Thomas Neumann. Cardinality estimation done right: Index-based join sampling. In *CIDR*, 2017.

[40] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. Wander join: Online aggregation via random walks. In *SIGMOD*, 2016.

[41] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. Wander join and XDB: online aggregation via random walks. *TODS*, 2019.

[42] Gang Luo, Curt J Ellmann, Peter J Haas, and Jeffrey F Naughton. A scalable hash ripple join algorithm. In *SIGMOD*, 2002.

[43] Barzan Mozafari. Approximate query engines: Commercial challenges and research opportunities. In *SIGMOD Keynote*, 2017.

[44] Barzan Mozafari, Eugene Zhen Ye Goh, and Dong Young Yoon. CliffGuard: A principled framework for finding robust database designs. In *SIGMOD*, 2015.

[45] Barzan Mozafari and Ning Niu. A handbook for building an approximate query engine. *IEEE Data Eng. Bull.*, 2015.

[46] Barzan Mozafari, Jags Ramnarayan, Sudhir Menon, Yogesh Mahajan, Soubhik Chakraborty, Hemant Bhanawat, and Kishor Bachhav. SnappyData: A unified cluster for streaming, transactions, and interactive analytics. In *CIDR*, 2017.

[47] Rasmus Pagh, Morten Stöckel, and David P. Woodruff. Is min-wise hashing optimal for summarizing set intersection? In *PODS*, 2014.

[48] Niketan Pansare, Vinayak R. Borkar, Chris Jermaine, and Tyson Condie. Online aggregation for large mapreduce jobs. *PVLDB*, 4, 2011.

[49] Yongjoo Park, Michael Cafarella, and Barzan Mozafari. Visualization-aware sampling for very large databases. *ICDE*, 2016.

[50] Yongjoo Park, Barzan Mozafari, Joseph Sorenson, and Junhao Wang. VerdictDB: universalizing approximate query processing. In *SIGMOD*, 2018.

[51] Yongjoo Park, Ahmad Shahab Tajik, Michael Cafarella, and Barzan Mozafari. Database Learning: Towards a database that becomes smarter every time. In *SIGMOD*, 2017.

[52] Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. QuickSel: Quick selectivity learning with mixture models. *CoRR*, abs/1812.10568, 2018.

[53] Theoni Pitoura and Peter Triantafillou. Self-join size estimation in large-scale distributed data systems. In *ICDE*, 2008.

[54] Chengjie Qin and Florin Rusu. Pf-ola: a high-performance framework for parallel online aggregation. *Distributed and Parallel Databases*, 2013.

[55] Sajjadur Rahman, Maryam Aliakbarpour, Hidy Kong, Eric Blais, Karrie Karahalios, Aditya G. Parameswaran, and Ronitt Rubinfeld. I've seen "enough": Incrementally improving visualizations to support rapid decision making. *PVLDB*, 2017.

[56] Hong Su, Mohamed Zait, Vladimir Barrière, Joseph Torres, and Andre Menck. Approximate aggregates in oracle 12c, 2016.

[57] Arun Swami and K Bernhard Schiefer. On the estimation of join result sizes. In *EDBT*, 1994.

[58] David Vengerov, Andre Cavalheiro Menck, Mohamed Zaït, and Sunil Chakkappen. Join size estimation subject to filter conditions. *PVLDB*, 2015.

[59] Sai Wu, Beng Chin Ooi, and Kian-Lee Tan. Continuous sampling for online aggregation over multiple queries. In *SIGMOD*, 2010.

[60] Wentao Wu, Jeffrey F Naughton, and Harneet Singh. Sampling-based query re-optimization. In *SIGMOD*, 2016.

[61] Dong Young Yoon, Mosharaf Chowdhury, and Barzan Mozafari. Distributed lock management with rdma: Decentralization without starvation. In *SIGMOD*, 2018.

[62] Kai Zeng, Shi Gao, Barzan Mozafari, and Carlo Zaniolo. The analytical bootstrap: a new method for fast error estimation in approximate query processing. In *SIGMOD*, 2014.

[63] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. Random sampling over joins revisited. In *SIGMOD*, 2018.