An Agile Sample Maintenance Approach for Agile Analytics

Hanbing Zhang*†, Yazhong Zhang*†, Zhenying He*†, Yinan Jing*†, Kai Zhang*†, X. Sean Wang*†‡
{hbzhang17, zhangyz17, zhenying, jingyn, zhangk, xywangCS}@fudan.edu.cn
* School of Computer Science, Fudan University, Shanghai, China
† Shanghai Key Laboratory of Data Science, Shanghai, China
‡ Shanghai Institute of Intelligent Electronics and Systems, Shanghai, China

***Abstract*—Agile analytics can help organizations to gain and sustain a competitive advantage by making timely decisions. Approximate query processing (AQP) is one of the useful approaches in agile analytics, which facilitates fast queries on big data by leveraging a pre-computed sample. One problem such a sample faces is that when new data is being imported, re-sampling is most likely needed to keep the sample fresh and AQP results accurate enough. Re-sampling from scratch for every batch of new data, called the full re-sampling method and adopted by many existing AQP works, is obviously a very costly process, and a much quicker incremental sampling process, such as reservoir sampling, may be used to cover the newly arrived data. However, incremental update methods suffer from the fact that the sample size cannot be increased, which is a problem when the underlying data distribution dramatically changes and the sample needs to be enlarged to maintain the AQP accuracy. This paper proposes an *adaptive sample update* (ASU) approach that avoids re-sampling from scratch as much as possible by monitoring the data distribution, and uses instead an incremental update method before a re-sampling becomes necessary. The paper also proposes an enhanced approach (T-ASU), which tries to enlarge the sample size without re-sampling from scratch when a bit of query inaccuracy is tolerable to further reduce the sample update cost. These two approaches are integrated into a state-of-the-art AQP engine for an extensive experimental study. Experimental results on both real-world and synthetic datasets show that the two approaches are faster than the full re-sampling method while achieving almost the same AQP accuracy when the underlying data distribution continuously changes.**

I. Introduction

Making timely decisions is an important task for today's enterprises, in which data is continuously growing and changing. For example, on Alibaba's Singles' Day Shopping Festival, 812 million orders were placed within 24 hours on Nov. 11, 2017[1]. In such a scenario, the underlying data distribution will change significantly with order placement. It is difficult to make correct decisions if analytics relies only on historical data. Hence, there is an increasing demand for agile analytics [13], i.e., analytics that provides fresh and timely insights from not only large-scale historical base data but also up-to-date data. Agility requires (1) the ability to handle underlying data changes, and (2) the timely response to analytical queries.

Approximate query processing (AQP) is a candidate technique for the requirements of timely analytics [24]. Using the pre-computed samples, existing AQP works (e.g., [5]–[7], [16]) often provide approximate results in exchange for timely response. The sampling theory can guarantee confidence bounds on result accuracy [22]. Since many decisions often rely just on a "big picture", inaccurate probabilistic results are often tolerable.

However, existing AQP works mostly ignored the problem of the underlying data updates. With the coming new data, the pre-computed samples will become increasingly stale, rendering the query results increasingly unreliable. Consider a data warehouse (DW for short) that stores hourly Wikipedia page view statistics data from the wiki-page-view-statistics [1]. Assume that DW contains the most recent 24 hours' data, and a pre-computed sample with the 10% sampling rate has been constructed in advance. Assume also that during the next 12 hours, a new batch of data is added into the DW every hour. Consider the following query Q, which is to get the total page view count of the wiki pages that belong to the project 'kk'.

Q: **SELECT SUM**(*page_count*) **FROM** pagecounts
WHERE *project_name*='kk';

Fig. 1 shows the relative error (using the performance metrics in Section IV) of Q for the 12 hourly update points. The relative error is increasing along with the data updates since the incoming skewed data makes the initial sample increasingly stale to answer the query. Therefore, to guarantee the query accuracy, samples need to be maintained timely along with data changes.

Existing sample maintenance methods [23], [28] cannot quickly update the sample with a dynamically adjusted sample size that is used to guarantee the query accuracy when the distribution of the underlying data changes. For example, we can use a small sample to calculate the average value when the underlying data has a small variance, but when the incoming new data causes the variance to increase significantly, the sample of the same size will not be able to provide a result with the same accuracy guarantee. Hence, a larger sample

Zhenying He, Yinan Jing, and X. Sean Wang are the contact authors.

[1]https://www.forbes.com/sites/helenwang/2017/11/12/alibabas-singles-day-by-the-numbers-a-record-25-billion-haul/3c4da1951db1

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00071

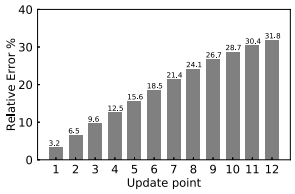


Fig. 1: Relative error of Q on the unchanged sample

is required. The natural sample update strategy called full re-sampling, is re-sampling from scratch to construct a new sample with the required sample size once a new batch comes. (It is worth mentioning that re-sampling in this paper differs in meaning from the term resampling used in the statistical theory.) Re-sampling is usually very expensive and adversely inhibits the agility of AQP. Another sample update strategy is incremental updating, such as reservoir sampling [28]. However, such a strategy can incrementally update the sample with a fixed sample size, but it cannot adjust the sample size while keeping the sample randomness. Thus, in AQP systems, to support agile analytics, maintaining samples at a lower cost while guaranteeing the query accuracy is a challenging and fundamental problem.

In this paper, we study an agile sample maintenance mechanism that adapts to data changes. The system will decide whether re-sampling is indeed needed when a new batch of data arrives by monitoring the data distribution. If the sample size does not need to be enlarged, the incremental update is employed. In this way, the sample maintenance cost can be reduced dramatically. We also develop a strategy, called *T-ASU*, to trade-off between query accuracy and sample maintenance speed. In summary, we make the following contributions with this paper:

- We introduce an *adaptive sample update* (ASU) approach that can choose between re-sampling and incremental sample update strategies according to the characteristics of existing historical data and new incoming data.
- We provide an enhanced approach T-ASU to speed up sample update further. As the experiment results indicate, T-ASU is faster than ASU in sample maintenance while sacrificing a bit of query accuracy.
- We integrate and evaluate ASU and T-ASU with a state-of-the-art AQP engine, and perform extensive experiments on real-world and synthetic datasets. Experimental results show that the two approaches are faster than the full re-sampling update strategy when the underlying data is continuously changing while achieving almost the same query accuracy as full re-sampling does.

The rest of this paper is organized as follows. In Section II, we formalize the problem and give an overview of an AQP system that includes our proposed agile sample maintenance strategies. Next, in Section III and Section IV, we describe our approaches and evaluate them using two different datasets. Finally, we review the related work in Section V and conclude the paper with Section VI.

II. Problem Definition and System Overview

A. Problem Definition

We call the data stored in the DW as *base data*. To deal with data updates, the DW system usually follows a batch data processing model shown in Fig. 2. The incoming new data are organized in batches (i.e., *batch data*) and are imported into the DW batch by batch.

As shown in Fig. 2, the *base data* at time t is denoted D_t. A sample S_t is constructed from D_t to support AQP. Next, at time $t+1$, a new batch of data B_{t+1} has come and the base data is changed to D_{t+1}, i.e., $D_{t+1} = D_t + B_{t+1}$[2]. Correspondingly, the sample needs to be updated from S_t to S_{t+1} to guarantee the query accuracy. A natural sample update strategy is to construct a new sample S_{t+1} from scratch to replace the previous sample S_t. Although such a new constructed sample is definitely good to guarantee the query accuracy, the update cost cannot be ignored, because every re-sampling needs to full scan the entire dataset. When the size of the base data is very large, one re-sampling operation will cost tens of minutes to several hours or even more. Obviously, too many expensive re-sampling operations will reduce the agility of the system. Therefore, it is not a wise choice for us to indiscriminately reconstruct a new sample for each batch. We need to be careful when maintaining the samples after the underlying data is updated.

The paper started with an observation that it would be unnecessary for us to re-sample from scratch if the underlying data has not changed dramatically after including a new batch. In such a situation, we may update the sample incrementally without changing the sample size instead of re-sampling to significantly reduce the sample update cost, while guaranteeing the query accuracy. Certainly, re-sampling is eventually unavoidable when the new batches of data dramatically change the underlying data distribution so a larger sample is required. Now the question is which update strategy, re-sampling or incremental updating, is the better choice for sample maintenance when the system receives a new batch of data. Hence, in this paper, the problem can be expressed as how to choose between the above two sample update strategies to minimize the sample update cost while guaranteeing the query accuracy. We define this problem as follows:

$$\begin{aligned} &\min cost(S_t \rightarrow S_{t+1}) \\ s.t. \quad &Pr[|\tilde{\theta}(S_{t+1}) - \theta(D_{t+1})| \leqslant \epsilon] \geqslant \alpha \end{aligned}$$

where:

- D_t and D_{t+1} are the *base data* at time t and $t+1$;
- S_t and S_{t+1} are the samples corresponding to D_t and D_{t+1}, respectively, and $cost(S_t \rightarrow S_{t+1})$ is the sample update cost from S_t to S_{t+1};
- $\tilde{\theta}(\cdot)$ is the approximate query result based on the sample, while $\theta(\cdot)$ is the exact query result based on the underlying data;

[2]Assume there are no duplicate records in these datasets and hence the "+" operation between datasets is equal to $\cup$ operation in this paper.

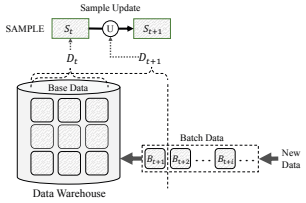


Fig. 2: Data processing model and problem definition

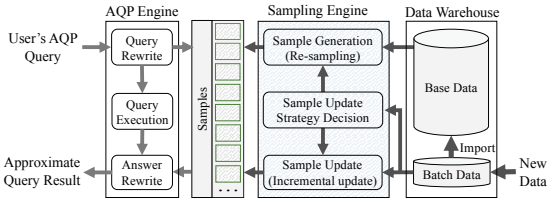


Fig. 3: AQP system with agile sample maintenance strategies

- ϵ is the error bound that is equal to the half-width of the confidence interval;
- α is the confidence level.

Specifically, for the grouping queries, $\tilde{\theta}(\cdot)$ and $\theta(\cdot)$ are two result sets. The ϵ and α can be set by users in their modified AQP queries [4] to make the query accuracy meet their needs.

B. System Overview

Fig. 3 gives an overview of an AQP system that integrates our agile sample maintenance strategies with a usual AQP engine. The system is composed of two parts: *AQP Engine* and *Sampling Engine*.

AQP Engine: The AQP engine is responsible for answering queries by leveraging on pre-computed samples. After a query is posed, the AQP engine firstly rewrites the query and execute it on the sample to get an intermediate result. Then, the engine needs to rewrite this intermediate result according to the sampling rate to obtain the approximate result with an error bound. Finally, the engine returns the approximate result. Since the query processing procedure of existing AQP engines is similar to each other, we can borrow from any one of them into our system.

Sampling Engine: The sampling engine is responsible for constructing and maintaining the samples. To tackle data changes, we implement a new sampling engine. Initially, the sampling engine constructs samples from the base data. When a new batch arrives, we evaluate the influence of the new batch on the current samples and decide whether we have to re-sample from scratch because of this new batch. If the new batch dramatically changes the underlying data distribution, re-sampling becomes inevitable, and the sampling engine will reconstruct a new sample of larger size from the updated dataset that includes the new batch. Otherwise, the sampling engine will update the sample incrementally in place using a strategy like reservoir sampling, to make it work well for queries. Compared with re-sampling, the incremental update does not need to scan all the data in DW and hence makes the whole system adapts to the data changes in a much more agile manner.

Through the collaboration between AQP engine and Sampling engine, the system can help analysts get fresh results from a large volume of historical and up-to-date data. Since the sampling engine is independent of the AQP engine, we can easily replace the sampling engine in the existing AQP systems with our engine proposed in this paper.

III. Methods

A. Sample Update Strategy Decision

For random sampling, a larger sample size generally leads to higher precision. Due to the storage constraint and the query processing speed requirement, the sample size is usually much smaller than the data size. Thus, query results on the sample are approximate with an error bound. When a new batch arrives, the original sample may become invalid to guaranteeing the specified accuracy since the new batch may change the distribution of underlying base data. Therefore, updating samples is required to adapt to the changes.

To update samples, there are many different update strategies. The simplest way to update the sample is by constructing a new sample to replace the old sample by re-sampling. However, it is unnecessary to re-sample when the expected size of the new sample does not need to be increased. To avoid too many expensive re-sampling operations, we can make a wiser choice in terms of the expected sample size. If the expected sample size is equal to or smaller than the old sample size, we can update the sample incrementally by taking an incremental random sampling method like reservoir sampling [28], instead of re-sampling from scratch. In such a scenario, we can regard the incoming batch data as a data stream and update the sample increasingly by applying reservoir sampling while keeping the sample size unchanged. In contrast, if the expected sample size is larger than the old sample size, we have to re-sample from scratch. Therefore, we need a condition to decide when we have to re-sample.

In terms of the user-specified accuracy requirement, we can deduce the expected sample size through the formula that calculates the error bound. For the given dataset D and aggregation function, we can calculate the error bound ϵ with the confidence level α in terms of the uniform sample S and its underlying base data D. Moreover, the calculation formula of ϵ only has a little difference between the different aggregation functions. Here, we take AVG as an example. The calculation formula for the error bound of AVG is shown as follows [22]:

$$\epsilon = z \cdot \sqrt{1 - \frac{|S|}{|D|}} \cdot \frac{\sigma}{\sqrt{|S|}} \tag{1}$$

, where the value of z corresponding to the given confidence level α can be figured out by looking up the standard normal distribution table in terms of probability theory [22], $|S|$ is the sample size, $|D|$ is the size of the base data, and σ is the standard deviation on the attribute that is involved in the AVG

function. Note that although α does not appear in Formula 1, Formula 1 still connects to α indirectly through z. Through Formula (1), we can easily deduce the sample size $|S|$ by Formula (2), which is rounded up to an integer.

$$|S| = \lceil \frac{z^2\sigma^2}{\epsilon^2 + \frac{z^2\sigma^2}{|D|}} \rceil \tag{2}$$

Specifically, in Formula (2), $|S|$ is monotonically increasing when σ^2 increases with the same $|D|$, ϵ and z.

Theorem 1. *For all dataset D of the same size $|D|$, user-specified confidence level α and error bound ϵ, $|S|$ in Formula* (2) *is monotonically increasing when σ^2 increases.*

Proof. Since the confidence level α is fixed, we have the same z in Formula (2) according to probability theory. For $\forall(\sigma_1^2 < \sigma_2^2)$ with the same $|D|$, ϵ and z, we get the following equations based on Formula (2)

$$|S_1| = \lceil \frac{z^2\sigma_1^2}{\epsilon^2 + \frac{z^2\sigma_1^2}{|D|}} \rceil \; and \; |S_2| = \lceil \frac{z^2\sigma_2^2}{\epsilon^2 + \frac{z^2\sigma_2^2}{|D|}} \rceil,$$

and the relation between the two sample size is $|S_1| \leq |S_2|$ because

$$\frac{z^2\sigma_1^2}{\epsilon^2 + \frac{z^2\sigma_1^2}{|D|}} - \frac{z^2\sigma_2^2}{\epsilon^2 + \frac{z^2\sigma_2^2}{|D|}} = \frac{z^2\epsilon^2|D|^2(\sigma_1^2 - \sigma_2^2)}{(\epsilon^2|D| + z^2\sigma_1^2)(\epsilon^2|D| + z^2\sigma_2^2)} < 0$$

Therefore, in Formula (2), $|S|$ is monotonically increasing for $\forall(\sigma_1^2 < \sigma_2^2)$ with the same $|D|$, ϵ and z. □

Moreover, in Formula (2), $|S|$ is also monotonically increasing for $\forall(|D_1| < |D_2|)$ with the same σ^2, ϵ and z. The new batch will need to increase the sample size even if the data distribution is not changed. Similar to Theorem 1, this fact can easily be proved. Thus, the sample size will increase if the variance increases or the data size increases. However, the sample size may not need to increase when the data size increases but the variance decreases. Hence, we can get the condition of re-sampling through the following Property 1. We can use Property 1, instead of the sample size shown in Formula 2, to determine whether re-sampling is needed.

Property 1. *Given a uniform sample S on the base data D, a new batch of data B that changes the variance σ^2 on one certain attribute/column by $\Delta\sigma^2$, i.e., $\Delta\sigma^2 = \sigma_{D+B}^2 - \sigma_D^2$. Given a user-specified confidence level α and error bound ϵ, there exists a bound δ that if $\Delta\sigma^2$ is larger than it, the sample must be re-sampled from scratch, otherwise, the sample can be updated incrementally.*

When a new batch B is added into the base data D, the data size will be changed from $|D|$ to $|D'| = |D| + |B|$. Let the change of variance σ^2 caused by the new batch be $\Delta\sigma^2$, i.e., $\Delta\sigma^2 = \sigma_{D+B}^2 - \sigma_D^2$. Then, the expected size of the new sample S' will become

$$|S'| = \lceil \frac{z^2(\sigma^2 + \Delta\sigma^2)}{\epsilon^2 + \frac{z^2(\sigma^2 + \Delta\sigma^2)}{|D'|}} \rceil. \tag{3}$$

Note that $\Delta\sigma^2$ can be calculated incrementally [15] when the new batch is added into the base data, instead of been calculated by scanning the whole dataset. If the expected sample size is larger than the original sample size, i.e., $|S'| > |S|$, we have to re-sample from scratch. Otherwise, we can incrementally update the sample. Eventually, we get the bound δ and the re-sampling condition as follows:

$$\begin{aligned} |S'| > |S| &\Rightarrow \lceil \frac{z^2(\sigma^2 + \Delta\sigma^2)}{\epsilon^2 + \frac{z^2(\sigma^2 + \Delta\sigma^2)}{|D'|}} \rceil > |S| \\ &\Rightarrow \Delta\sigma^2 > \frac{\epsilon^2 \cdot |D'| \cdot |S| - z^2 \cdot \sigma^2 \cdot (|D'| - |S|)}{z^2 \cdot (|D'| - |S|)} = \delta \end{aligned} \tag{4}$$

Therefore, we can decide whether to re-sample by judging whether $\Delta\sigma^2$ is larger than the bound δ.

The re-sampling condition on other aggregation functions can be achieved in a similar way to the AVG function. The difference between them is Formula (1) (e.g., the error bound calculation in the SUM function should multiply $|D|$ on the right-hand side of Formula (1)). In addition, according to different attributes/columns, we will get many different δ for multiple aggregation functions, since the variance σ^2 on different attribute is different. Therefore, in practice, we use the minimum value of multiple δ as the lower bound to determine re-sampling finally. For stratified samples, we can also use the judgment condition on the uniform sample to make a re-sampling decision, because a stratified sample can be constructed by relying on a uniform sample in practice.

B. Adaptive Sample Update (ASU) Algorithm

According to the Property 1, we propose an *adaptive sample update* (ASU) algorithm which can adapt to the data changes continuously. ASU firstly determines whether the sample size needs to increase and then chooses corresponding update strategies to update samples. If the sample size needs to increase, the re-sampling will be used to update the sample. Otherwise, the sample will be updated by an incremental sample update method that is faster than re-sampling.

The ASU works as Algorithm 1 shown. The input of ASU includes the base data D, the sample S on the base data, the sample type T_S (including uniform sample and stratified sample), the feature information F_D about the base data, a new batch B, the user-specified confidence levels $\{\alpha_1, \alpha_2, ...\}$ and error bounds $\{\epsilon_1, \epsilon_2, ...\}$ for different aggregations. Specifically, the information F_D contains the statistical data $F_D.S$ (e.g., $F_D.S_{\sigma^2}$ is the variances for different attributes on base data D), stratum information $F_D.L$ (e.g., strata's names and sizes) and stratified sampling rule $F_D.R$ (e.g., equal allocation and proportional allocation).

Algorithm 1: Adaptive sample update (ASU)

Input: base data D, sample S, sample type T_S, data feature F_D, new batch B, confidence levels $\{\alpha_1, \alpha_2, ...\}$, error bounds $\{\epsilon_1, \epsilon_2, ...\}$.
Output: updated sample S'.

1 F_B = feature statistic on batch B;
2 F' = incremental update feature F_D by F_B;
3 $\Delta\sigma^2 = F'.S_{\sigma^2} - F_D.S_{\sigma^2}$;
4 δ = calculate the minimum bound with $F_D.S$, $|S|$, $|D'| = |D| + |B|$, $\{\alpha_1, \alpha_2, ...\}$ and $\{\epsilon_1, \epsilon_2, ...\}$;
5 **if** $\Delta\sigma^2 > \delta$ **then**
6 n = calculate the maximum sample size with $F'.S$, $|D'| = |D| + |B|$, $\{\alpha_1, \alpha_2, ...\}$ and $\{\epsilon_1, \epsilon_2, ...\}$;
7 S' = select n tuples from $D + B$ according to T_S (re-sampling);
8 **else**
9 **if** $T_S ==$ *uniform sample* **then**
10 S' = call conventional reservoir sampling to update S with B;
11 **else if** $T_S ==$ *stratified sample* **then**
12 $S' = \emptyset$;
13 **foreach** *stratum* $l_i \in F'.L$ **do**
14 n_{l_i} = allocate sample size to l_i based on $|S|$ and previous sample rule $F_D.R$;
15 B_{l_i} = read the tuples of l_i from B;
16 **if** $l_i \in F_D.L$ **then**
17 S_{l_i} = read the tuples of l_i from S;
18 S'_{l_i} = call **Algorithm 2** to update S_{l_i} with $F_D.L_{l_i}, B_{l_i}$ and n_{l_i};
19 **else**
20 S'_{l_i} = random select n_{l_i} tuples from B_{l_i};
21 $S' = S' + S'_{l_i}$;
22 return S';

Algorithm 2: Stratum sample update

Input: sub-base data size $|D_{l_i}|$, sub-sample S_{l_i}, sub-batch B_{l_i}, expected sub-sample size n_{l_i}.
Output: updated sample S'_{l_i}.

1 $S'_{l_i} = \emptyset$;
2 **if** $n_{l_i} > |S_{l_i}|$ **then**
3 $k = |S_{l_i}|$, $S'_{l_i} = S_{l_i}$;
4 **foreach** *tuple* $t \in B_{l_i}$ **do**
5 $k = k + 1$;
6 **if** $k \leqslant n_{l_i}$ **then**
7 $S'_{l_i} \leftarrow t$;
8 **else**
9 S'_{l_i} = call conventional reservoir sampling to update S'_{l_i} with t;
10 **else**
11 S_r = random select $|n_{l_i}|$ tuples from S_{l_i} to retain;
12 S'_{l_i} = call conventional reservoir sampling to update S_r with B_{l_i};
13 return S'_{l_i};

With these inputs, ASU first incrementally updates the data feature when the new batch is added to the base data (line 1-2) and decides whether the sample size will be affected by the data changes (line 3-5). Note that we use the minimum bound δ to meet the user's requirements for different aggregation functions as we discussed in Section III.A. Then, if the changes of data distribution $\Delta\sigma^2$ is larger than the bound δ, ASU re-samples from the base data and batch data according to the sample type (line 7). The calculation of the expected sample size is based on the data size, data distribution, confidence level and error bound (line 6), and it is different for various aggregation functions. Otherwise, if the sample size is not affected by the data changes, ASU updates samples incrementally. For a uniform sample, the conventional reservoir sampling [28] can be used to update it directly (line 10). For a stratified sample, the new data will change the previous data feature and affect the sample generation. Therefore, we stratify the sample and the batch data based on the stratified rule on the base data, and update the sample in each stratum independently (line 13-21). If a stratum exists in the base data, Algorithm 2 is used to update it (line 16-18). Otherwise, it can be updated by the random sampling (line 20). The $F_D.L_{l_i}$ (line 18) is the sub-base data size of stratum l_i. Specifically, we use a prefix 'sub' to represent one stratum in the whole data, e.g., sub-base data size $|D_{l_i}|$ is the data size of stratum l_i in base data D. Next, we use an example to explain how to use ASU to update a stratified sample incrementally when the re-sampling is not needed.

Example. *Given a dataset D ($|D| = 6000$) which can be divided into two strata l_1 ($|D_{l_1}| = 100$) and l_2 ($|D_{l_2}| = 5900$) according to the specified attribute and the sample size is 600. Since the size of stratum l_1 is less than 300, we can only extract 100 tuples from D_{l_1} and extract 500 tuples from D_{l_2} to satisfy the required sample size. These sub-samples are called S_{l_1} ($|S_{l_1}| = 100$) and S_{l_2} ($|S_{l_2}| = 500$). When a new batch B ($|B| = 4000$) is added into the base data, if $\Delta\sigma^2$ is less than the bound δ, the sample size does not need to increase. Assume batch B has 900 tuples of stratum l_1 (B_{l_1}), 100 tuples of stratum l_2 (B_{l_2}) and 3000 tuples of stratum l_3 (B_{l_3}). Therefore, the updated dataset D' is composed of 1000 tuples of stratum l_1 ($|D'_{l_1}| = 1000$), 6000 tuples of stratum l_2 ($|D'_{l_3}| = 6000$) and 3000 tuples of stratum l_3 ($|D'_{l_3}| = 3000$). Since the sample size does not increase, we extract 200 tuples from each stratum in D' based on the equal allocation rule. For stratum l_1, the sub-sample size $|S'_{l_1}|$ will increase from 100 to 200 because the sub-data size increases from 100 to 1000. Hence, we first add 100 tuples from B_{l_1} into S_{l_1} and then use the remaining tuples in B_{l_1} to update S_{l_1} according to the reservoir sampling algorithm (line 1-9 in Algorithm 2). For stratum l_2, the sub-sample size $|S'_{l_2}|$ will decrease from*

500 to 200. Hence, we first randomly evict 300 tuples from S_{l_2} and then use the tuples in B_{l_2} to update S_{l_2} according to the reservoir sampling algorithm (line 10-12 in Algorithm 2). For stratum l_3, since it is a new stratum, we extract 200 tuples from B_{l_3} randomly to construct a new sub-sample S'_{l_3} (line 20 in Algorithm 1). Finally, we get an updated stratified sample S' with 600 tuples.

To reduce the number of re-samplings, we construct a sample that can meet a higher confidence level (*loose confidence level*) than the user-specified confidence level when re-sampling, and hence the new sample size is larger than what we calculated. For example, if the confidence level set by users is 95%, ASU will create a sample using the 96% loose confidence level when performing re-sampling. Therefore, the size of the new sample is 1.1 times larger than that of 95% confidence level, and it can reduce the number of re-sampling in the subsequent data updates.

The I/O cost of sample update in ASU for the base data D and new batch B is $|D| + |B|$ if it needs re-sample from scratch. Otherwise, the I/O cost is $r \cdot |D| + |B|$, where r is the sampling rate. Therefore, set the probability that the sample size does not need increase is β, the expected I/O cost in ASU is $(1 - (1 - r) \cdot \beta) \cdot |D| + |B|$.

Workload: In Algorithm 1, we use the minimum bound δ which is calculated by the global statistics (variance and data size) to decide whether the sample is needed to be reconstructed, and the maximum expected sample size is also calculated by the global statistics. However, for some special queries, the sample update by ASU cannot guarantee their accuracy, because the sample size assigned to the query related subset is not large enough to construct a good sample. The insufficient sample size on the subset is caused by the inconsistency of data distribution between the subset and the global dataset (underlying data), i.e., the sample size calculated by the statistics on subset is larger than the sample size assigned by the global sample according to the data size of the subset. It's similar to the cold start problem that we don't know which subset the user is interested in and we cannot enumerate all possible subsets of the underlying data due to the expensive time cost. Thus, in order to improve the query accuracy on the updated sample, we exploit the workload (query history on the underlying data) to identify the queries which are important to users and decide the sample update strategy according to the distribution changes of the important subsets.

The workload is composed of a variety of queries. The duration of query executions can be divided into several time slots $1, 2, ..., T$. For a query Q_i, it may be executed many times in one time slot. The score of Q_i will be calculated by the following steps:

Step 1: Calculate the scores of the query Q_i in sequential time slots, $QS_{i,1}, QS_{i,2}, ..., QS_{i,T}$. The score in the time slot t, $QS_{i,t}$, equals to the number of occurrences of Q_i in this slot.

Step 2: According to a decay function [19], [30], the query scores will be reduced by the chronological order of time slots, i.e., $QS_{i,t} = w_t * QS_{i,t}$, where w_t is a decay weight.

Step 3: If the query is executed periodically, the query score will be enlarged by a factor p ($p > 1$). The final score of Q_i is calculated by Formula (5).

$$QS_i = p * \sum_{t=1}^{T} w_t * QS_{i,t} \tag{5}$$

Then, for those queries with top-k scores, the local distribution of the subsets covered by those queries will be calculated. In terms of the local distribution, the expected sample size will be adjusted, except for considering global data distribution. The adjusted sample will benefit the query accuracy of those known queries in the historical workload. Furthermore, to adapt to the workload changes, we periodically update the query score. If there is no historical workload available, ASU still works well by using the global data distribution to decide whether re-sampling is needed.

C. T-ASU Algorithm

The re-sampling in ASU is used to ensure that the updated sample can provide query results of high quality when the data distribution changes dramatically. However, for a large dataset, ASU may be still too expensive due to the inevitable re-sampling operations. Therefore, to further speed up sample update, we propose the *T-ASU*, which makes a trade-off between the query accuracy and the sample update speed, i.e., to reduce the number of re-samplings by sacrificing accuracy.

In T-ASU, we replace the re-sampling in ASU with the Algorithm 3 that exploits the *adaptive reservoir sampling* (ARS) [8]–[10] to update the enlarged sample incrementally by sacrificing a little query accuracy. ARS requires a minimum number of new tuples *m* to maintain the *uniformity confidence* (UC) of sample exceeding a given threshold ζ, which represents the sample randomness. The UC is calculated as follows:

$$UC = \frac{\sum_{x=\max(0,n-m)}^{|S|} \binom{|D|}{x}\binom{m}{n-x}}{\binom{|D|+m}{n}} \tag{6}$$

, where $|D|$ is the underlying data size, $|S|$ is the sample size, n is the expected sample size and m is the number of incoming tuples. Thus, the value of minimum m can be derived from this formula. Intuitively, the higher uniformity confidence is, the higher sample quality (query accuracy) can be.

However, for the continuous use of ARS (no re-sampling occurs between them), it will make the calculated value of uniformity confidence larger than the actual value of uniformity confidence and make a misjudgment that the uniformity confidence is larger than the given threshold. As a result, if we use ARS to update the sample that indeed should be re-sampled, the query result will be more inaccurate than user-specified. The larger uniformity confidence is caused by the larger value of $\binom{|D|}{x}$ in Formula (6). To facilitate the understanding of this situation, we provide an example below.

Example. *Given a dataset D ($|D| = 100$), a uniform sample S ($|S| = 10$) on D, the given uniformity confidence threshold*

Algorithm 3: Sample expanding algorithm

Input: base data D_t, sample S_t, sample type T_S, data feature F_D and F', new batch B_t, expected sample size n_t, uniformity confidence threshold ζ, factor u_{t-1}.

Output: updated sample S'

$\hat{m}_t$ = set the upper bound of minimum m_t according to $|D_t|$, $|S_t|$, $|B_t|$, n_t and ζ;
$\hat{m}'_t = \mu_{t-1} \cdot \hat{m}_t$;
if $\hat{m}'_t \leqslant |B|$ **then**
  if T_S == *uniform sample* **then**
    S'_t = call **Algorithm 4** to update S_t with $|D_t|$, B_t and n_t;
  else if T_S == *stratified sample* **then**
    $S'_t = \emptyset$;
    foreach *stratum* $l_i \in F'.L$ **do**
      n_{l_i} = allocate sample size to l_i based on n_t and previous sample rule $F_D.R$;
      B_{l_i} = read the tuples of l_i from B_t;
      if $l_i \in F_D.L$ **then**
        S_{l_i} = read the tuples of l_i from S;
        S'_{l_i} = call **Algorithm 4** to update S_{l_i} with $F_D.L_{l_i}, B_{l_i}$ and n_{l_i};
      else
        S'_{l_i} = random select n_{l_i} tuples from B_{l_i};
      $S'_t = S'_t + S'_{l_i}$;
  μ_t = calculate the factor (using Formula (9)) with $|D_t|$, $|B_t|$,$\hat{m}_t$ and μ_{t-1};
else
  S'_t = select n_t tuples from $D_t + B_t$ according to T_S (re-sampling);
  $\mu_t = 1$;
return S'_t, μ_t;

Algorithm 4: Adaptive reservoir sampling

Input: base data size $|D|$, sample S, data stream $d_{1,2,...}$, number of incoming tuples m, expected sample size n.

Output: updated sample S'

if $n > |S|$ **then**
  S_r = random select x tuples from S to retain, where $\max\{0, n-m\} \leqslant x \leqslant |S|$;
  S_c = random select n-x tuples from $d_{1,...,m}$;
  $S_{rc} = S_r + S_c$;
  S' = call conventional reservoir sampling to update S_{rc} with $d_{m+1,...}$;
else
  S_r = random select $|n|$ tuples from S to retain;
  S' = call conventional reservoir sampling to update S_r with $d_{1,2,...}$;
return S';

$\zeta = 95\%$*, the user-specified confidence levels and error bounds. There is a new batch* B_1 *(*$|B_1| = 40$*) added into the dataset* D *at time* t_1 *that makes the change of data distribution larger than the bound (Property 1) and the expected sample size is 11. Since the calculated uniformity confidence 97.9% (using Formula* (6) *with* $m = |B_1|$*) is larger than the given threshold* ζ*, we can use the ARS to update the sample* S*. The updated sample* S' *(*$|S'| = 11$*) is composed by 7 tuples (randomly pick) from dataset* D *and 4 tuples from batch* B_1*.*

Then, at time t_2*, a new batch* B_2 *(*$|B_2| = 40$*) is added into the updated dataset* D' *(*$|D'| = |D| + |B_1| = 140$*) that also makes the change of data distribution larger than the bound (Property 1) and the expected sample size is 12. According to Formula* (6)*, the calculated uniformity confidence 95.6% (*$m = |B_2|$*) is larger than the given threshold* ζ *and hence we can use ARS to update the sample* S'*. However, the actual value of uniformity confidence is 59% at this moment. In the calculated uniformity confidence, the calculated* $\binom{|D|}{x}$ *is* $\binom{|D'|=140}{x}$*, which means the* x *tuples is uniform sampling from the base data* D'*. But in fact, the actual* $\binom{|D|}{x}$ *is* $\sum_{i=\max(0,x-4)}^{\min(7,x)} \binom{|D|=100}{i}\binom{|B_1|=40}{x-i}$ *for each* x *in range* $\{0, 11\}$*, where 7 and 4 are the numbers of tuples extract from* D *and* B_1 *respectively. As the calculated* $\binom{|D|}{x}$ *is larger than the actual* $\binom{|D|}{x}$*, the calculated uniformity confidence becomes larger than the actual uniformity confidence, and hence we have a wrong decision that the sample* S' *can be updated by the ARS.*

Thus, to get an accurate uniformity confidence when ARS is executed continuously, we change the $\binom{|D|}{x}$ in Formula (6) as follows:

$$\binom{|D|}{x} = \binom{|D_t|}{x_t} = \sum_{a=max(0,x_t-x_{t-1})}^{min(x_t,x_{t-1})} \binom{|D_{t-1}|}{a}\binom{|B_{t-1}|}{x_t-a} \tag{7}$$

, where t is the time that sample will be updated, $t-1$ is the previous time that sample is updated by ARS, $|D_t|$ and $|D_{t-1}|$ are the base data size at time t and $t-1$, x_t and x_{t-1} are the numbers of tuples extract from the data D_t and D_{t-1}. Then, we use Formula (8) to calculate the value of minimum m_t, which makes the uniformity confidence larger than the given threshold ζ at time t when ARS is executed continuously, where n_t is the expected sample size.

$$\arg\min_{m_t} \frac{\sum_{x_t=\max(0,n-m)}^{|S|}\binom{|D_t|}{x_t}\binom{m_t}{n_t-x_t}}{\binom{|D_t|+m_t}{n_t}} \geq \zeta \tag{8}$$

Moreover, since the value of minimum m_t cannot be timely calculated on the large data size, we set the value of m_t empirically to decide whether we can use ARS to update the enlarged sample. Further, we enlarge the upper bound of m_t with a factor μ_{t-1} to ensure that the Algorithm 4 is correctly

called by the Algorithm 3 at time t. The factor μ_t is calculated by Formula (9), where the $\hat{m}_t$ is the upper bound of m_t.

$$\mu_t = \mu_{t-1} + \log_{\hat{m}_t} \frac{|D_t| + |B_t|}{|B_t| - \hat{m}_t + 1} \tag{9}$$

The default value of μ_t is 1, i.e., $\mu_0 = 1$, and it will be set to 1 when the sample is updated by re-sampling because the uniformity confidence of re-sampling is 100%. In Formula (9), we use the logarithmic function to approximate the uniformity confidence because the value of uniformity confidence with the m increment is similar to this logarithmic function. It also meets the rule that a larger uniformity confidence at time $t-1$ will make the minimum m_t smaller.

The Algorithm 3 works as follows. Firstly, we get the upper bound $\hat{m}_t$ of the minimum size of new tuples m_t empirically (line 1). Then, we enlarge the $\hat{m}_t$ (line 2) to ensure that the Algorithm 4 (ARS) is correctly called. If the batch size is larger than the $\hat{m}'_t$ (line 3), we call Algorithm 4 to update the sample (line 4-16). Otherwise, we re-sample from the base data and batch data according to the sample type (line 19). Note that we use the $|B_t|$ instead of the $\hat{m}'_t$ in Algorithm 4 (line 5 and line 13). After the sample update, we calculate the value of factor μ_t. For the sample updated by Algorithm 4, we use Formula (9) to calculate μ_t (line 17). For re-sampling, we set μ_t to 1 (line 20) because the uniformity confidence of re-sampling is 100%. Therefore, the I/O cost of sample update in T-ASU depends on the user-specified uniformity confidence threshold (e.g., 95%) and the size of the new batch.

IV. Experiments

A. Experiment setup

We implement *full re-sampling* (FULL), *adaptive sample update* (ASU) and T-ASU strategies, and integrate our sampling engine into Verdict [26]. All experiments are conducted on a Spark cluster with 10 nodes (each node with Intel Xeon E5-2620, 64GB RAM, and 1.77TB HDD disk under Ubuntu Linux 14.04.1 LTS).

Performance Metrics: Two metrics are used: (1) the time cost of sample updates (abbr. *time cost*), which is the time used to update the sample when a new batch is added into the base data; (2) the *relative error* (*RE*), which is used to evaluate the query accuracy. The relative error is calculated by Formula (10), where $\hat{\theta}(S_t)$ is the approximate query result based on the sample S_t and $\theta(D_t)$ is the query result based on the underlying data D_t at time t. Specifically, for the grouping queries, the relative error is the average value upon each group.

$$RE = \frac{|\hat{\theta}(S_t) - \theta(D_t)|}{\theta(D_t)} \tag{10}$$

In the following experiments, the relative error is the average value of ten separate experiments with the same experimental settings. By default, the confidence level is set at 95%.

Real dataset (Wiki): We use the wiki-page-view-statistics [1], containing the hourly global page-view statistics of the Wiki media projects. Each tuple in the dataset is composed of four attributes: *page_name*, the title of the page, *page_count*, the number of requests for the page in the hour, *page_size*, the size of the requested page, and *project_name*, the project to which the page belongs. We extract the data from January 1, 2016, to January 21, 2016. The total data size is 75GB. Since the dataset is organized in sequential hours, we naturally divide the data into batches by hours. By default, we treat the first 24 hours' data as *base data*, and let the size of *batch data* be 1-hour. We call such a base data update pattern as 24-1.

To evaluate the effectiveness of different sample update strategies, we use the following four typical example queries.

Q_1: **SELECT SUM**(*page_count*) **FROM** pagecounts **WHERE** *project_name*='kk';

Q_2: **SELECT COUNT**(*) **FROM** pagecounts **WHERE** *project_name*='www';

Q_3: **SELECT AVG**(*page_count*) **FROM** pagecounts **WHERE** *page_size*<7000;

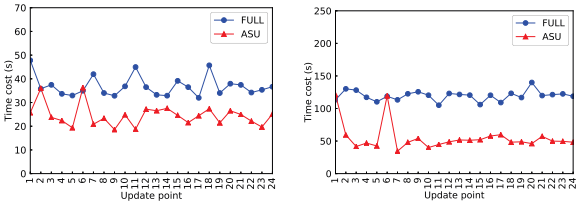
Q_4: **SELECT** *project_name*, **SUM**(*page_count*) **FROM** pagecounts **GROUP BY** *project_name*;

The above four queries cover various aggregation functions, conditions and different data subpopulations, e.g., the data with *project_name*='www' for Q_2 is a rare subpopulation.

Synthetic dataset (TPC-H): We generate 450GB TPC-H data. We then use a 350GB subset of the data as the *base data* and divide the remaining data equally into 20 batches. We denote this update pattern 350-5 for short. To evaluate the query accuracy, we select four typical queries from the TPC-H benchmark: a simple query Q6, two join queries Q14, Q19, and a grouping query Q1 (denoted as Q_I, Q_{II}, Q_{III} and Q_{IV}, respectively).

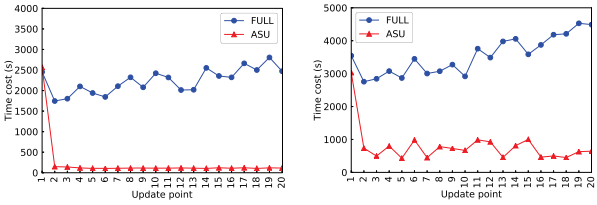
B. Experiment results

Performance on the Wiki dataset (small dataset): We compare the performance between FULL and ASU on the Wiki dataset. We use the default 24-1 update pattern and the 10% initial sampling rate. Note that in the different aggregation functions and attributions, the user-specified error bound that has the minimum bound δ is 1.2 on the AVG(*page_count*). Fig. 4 shows the time cost of updating the uniform sample and the stratified sample on the 24 update points. For the uniform sample, the average time cost of ASU is about 68% of FULL (ASU 25s vs. FULL 37s). As shown in Fig. 4(a), ASU just performs re-sampling twice, namely after the second and sixth update batches to guarantee query accuracy since a significant change of the data distribution happens only after each of these two batches. For the stratified sample, the average time cost of ASU is about 46% of FULL (ASU 55s vs. FULL 119s). As shown in Fig. 4(b), ASU only performs re-sampling twice, namely after the first and sixth update batches. Due to extra computation, updating the stratified samples costs more time than that for a uniform sample. We evaluate the example queries on 24 update batches and obtain the average relative error. Fig. 5 shows the relative error of the four example queries with FULL and ASU. As shown in Fig. 5, ASU basically achieves the same query accuracy as FULL does. Both FULL and ASU can guarantee the query accuracy of Q_1, Q_2 and Q_4. For Q_3, the relative error of ASU is a little



(a) Uniform sample (b) Stratified sample

Fig. 4: The sample update cost on the Wiki (24-1)



(a) Uniform sample (b) Stratified sample

Fig. 6: The sample update cost on the TPC-H (350-5)

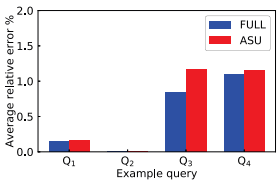


Fig. 5: The relative error on the Wiki (24-1)

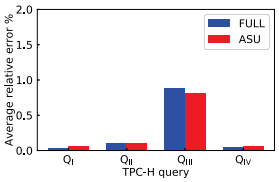


Fig. 7: The relative error on the TPC-H (350-5)

bit higher than that of FULL, because some of the tuples in the result of Q_3 are replaced by new tuples, which causes an accidental error. Even so, the relative error is still within the user-specified error bound.

Therefore, ASU is about 1.5x-2.2x faster than FULL in sample update speed while achieving almost the same query accuracy as FULL does on the four example queries.

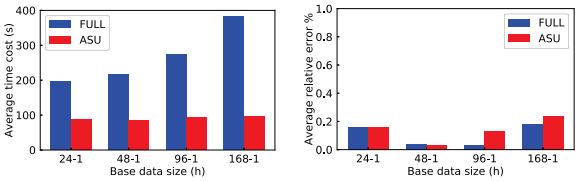
Performance on the TPC-H dataset (large dataset): We compare the performance between FULL and ASU on the larger TPC-H dataset. Here, we use the default 350-5 update pattern. Since the TPC-H dataset is much larger than the Wiki dataset, to get the query result quickly and meet the constraint of the memory size at the same time, we use a smaller initial sampling rate (1%) here. In the different aggregation functions and attributions, the user-specified error bound that has the minimum bound δ is 2.5 on the AVG(*l_extendedprice*). Fig. 6 shows the time cost of updating the uniform sample and the stratified sample on the 20 update points. As a result of the increased data size, the sample update speed of ASU is much faster than that of FULL. For the uniform sample, the average time cost of ASU is about 10.7% of FULL (ASU. 239s vs. FULL. 2242s). For the stratified sample, the average time cost of ASU is about 22.5% of FULL (ASU. 799s vs. FULL. 3548s). Note that ASU has only one re-sampling on the first update point, because the data distribution of the TPC-H dataset is uniform and it rarely changes. We evaluate the queries Q_I, Q_{II}, Q_{III}, Q_{IV} on 20 update points and get the average relative error. Theoretically, the relative error of FULL should be smaller than that of ASU. However, as shown in Fig. 7, the relative errors of FULL and ASU vary for different queries. Taking Q_{III} as an example, the average relative error of ASU is smaller than that of FULL due to the sample randomness. However, overall, ASU achieves almost the same query accuracy as FULL does.

In summary, ASU is about 4.4x-9.4x faster than FULL in sample update speed, while achieving almost the same query accuracy as FULL does on the four benchmark queries.

Effect of base data size: Based on the Wiki dataset, we fix the batch size as 1 hour and adjust the base data size as follows: 24 hours (24-1), 48 hours (48-1), 96 hours (96-1), 168 hours (168-1). Since the sample size will affect the sample update cost, to get comparable experimental results on base data of different sizes, we use 10%, 5%, 2%, and 1% initial sampling rate respectively to get samples of similar sizes.

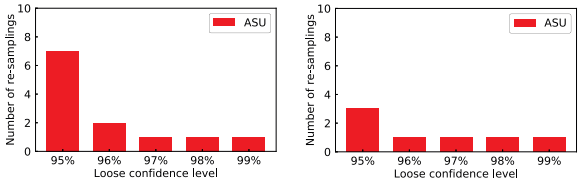
Fig. 8(a) shows the average time cost of 24 sample updates on various base data sizes. The average time cost of FULL increases significantly as the base data size increases since the re-sampling needs to scan the entire dataset. We also find that the base data size has a limited effect on the time cost of ASU because it can reduce the unnecessary re-sampling operations and update the sample incrementally. To evaluate the query accuracy on the above-mentioned experimental settings, we use the example query Q_1 to conduct experiments. As shown in Fig. 8(b), ASU achieves almost the same relative error as FULL. Although the relative error of ASU on 96-1 is higher than that of FULL, the relative error is still within the user-specified error bound. Therefore, the base data size has a small effect on the performance of ASU, but FULL cannot ignore the negative effect of the base data size.

Effect of batch data size: On the Wiki dataset, we fix the base data size as 168 hours and adjust the batch size as follows: 1 hour (168-1), 6 hours (168-6), 12 hours (168-12), 24 hours (168-24). The initial sampling rate is 1%. Fig. 9(a) shows the average time cost of 14 sample updates on various batch data sizes. The average time cost of FULL increases significantly as the batch data size increases, since the re-sampling needs to scan the entire dataset. We also find that the average time cost of ASU will be affected by the batch data size, because it needs to read the batch data to update the samples. However, the effect of batch data size on ASU is much less than that on FULL. To evaluate the query accuracy on the above-mentioned experimental settings, we use the example query Q_1 to conduct



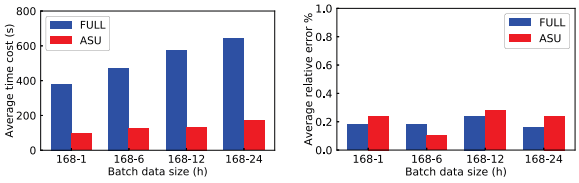
(a) Time cost (b) Query accuracy of Q_1

Fig. 8: The performance of FULL and ASU on the Wiki datasets (X-1) with different base data sizes



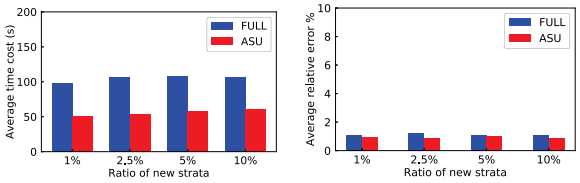
(a) 24-1 (b) 168-1

Fig. 10: The number of re-samplings in ASU on the Wiki datasets (24-1 and 168-1)



(a) Time cost (b) Query accuracy of Q_1

Fig. 9: The performance of FULL and ASU on the Wiki datasets (168-X) with different batch data sizes



(a) Time cost (b) Query accuracy of Q_I

Fig. 11: The performance of FULL and ASU on the TPC-H dataset with various ratios of new strata

experiments. As shown in Fig. 9(b), ASU achieves almost the same relative error as FULL. Therefore, the batch data size has only a small effect on the query accuracy of ASU, but it will slow down the sample update in both ASU and FULL.

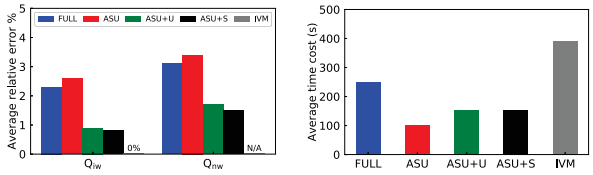
Effect of loose confidence level: We vary the loose confidence level from 95% to 99% and conduct experiments on Wiki w.r.t. 24-1 and 168-1. As shown in Fig. 10, the number of re-samplings decreases as the loose confidence level increases and become stable when the loose confidence level is high enough, such as the 96% loose confidence level in the 168-1 experiment. The number of re-samplings in 168-1 is less than in 24-1, because the data distribution on the large base data is more stable than that on the small base data when the new batch data arrives. Moreover, the sample on the loose confidence level does not take up a lot of extra storage space (e.g., the sample size on the 96% loose confidence level is about 1.1x times of 95% confidence level). Therefore, calculating the sample size with the loose confidence level can significantly reduce the number of re-samplings in ASU, while it will not increase storage overhead dramatically.

Effect of the new stratum: We evaluate the effect of the new stratum on the TPC-H. We construct an 11GB dataset that contains 10GB base data and 1GB batch data (denoted as 10-1). Based on the attribute *l_shipdate*, we fix the number of strata in the base data as 2000 and adjust the number of new strata in the batch data as follows: 20 (1%), 50 (2.5%), 100 (5%), 200 (10%). The initial sampling rate is 1% and there is only a stratified sample on the dataset. As shown in Fig. 11(a), the ratio of new strata has a limited effect on the time costs of both ASU and FULL. We use the queries Q_I to evaluate the query accuracy. As shown in Fig. 11(b), ASU achieves almost the same relative error as FULL, since they successfully extract tuples from the new strata to guarantee the query accuracy. Therefore, the ratio of new stratum has only a small effect on the performance of both ASU and FULL.

Effect of workload: We evaluate the effect of workload on both query accuracy and time cost on the TPC-H. Given the historical workload, ASU may benefit from the workload. As we know, the incremental view maintenance method (IVM) [31] can also benefit from the workload for the response time of known queries. Hence, we compare the performance of FULL, ASU, ASU with the aid of the workload, and IVM in this experiment. We construct two synthetic workloads with different distributions (Uniform and Skewed) and get two variants of ASU: ASU+U and ASU+S, respectively. Each workload is composed of 100 historical queries. The skewed workload is generated using a Zipf distribution with $z = 2$. Due to the space limitation, we just show the experiment results under this specific Zipf parameter.

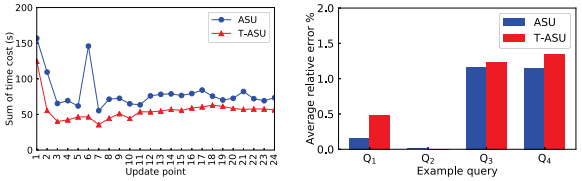
Fig. 12(a) shows the query accuracy of FULL, ASU, ASU+U, ASU+S, and IVM on the TPC-H. For a query, we consider two situations: (1) this query has appeared in the historical workload (denoted as Q_{iw}); (2) it has not appeared in the workload (denoted as Q_{nw}). For Q_{iw}, both ASU+U and ASU+S have a smaller relative error than ASU and FULL, because the samples adjusted by exploiting the workload can benefit the query accuracy of those known queries. Since Q_{iw} has appeared in the workload, the relative error of IVM is 0%. For Q_{nw}, the relative errors of ASU+U and ASU+S are also smaller than those of ASU and FULL. According to the workload used in the experiment, the sample size will be enlarged. A larger sample can also benefit the query accuracy of Q_{nw}, although Q_{nw} has not appeared in the historical workload. Since Q_{nw} has not appeared in the workload, IVM cannot answer the query Q_{nw} (denoted as N/A in Fig. 12(a)).

Fig. 12(b) shows the sample update cost of FULL, ASU,



(a) Query accuracy (b) Time cost

Fig. 12: The performance of FULL, ASU, ASU+U, ASU+S, and IVM on the TPC-H



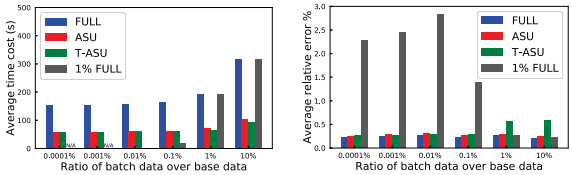
(a) Time cost (b) Query accuracy

Fig. 13: The performance of ASU and T-ASU on the Wiki dataset (24-1)

ASU+U, ASU+S, and the view maintenance cost of IVM. Among all of the approaches, ASU has the lowest time cost. Compared with ASU, on average, ASU+U and ASU+S will cost more time to maintain the samples, while they cost much less time than FULL. IVM costs much more time to maintain the materialized views than other approaches as new batches arrive. This is because the view maintenance cost mainly depends on the number of materialized views, and the number of views is proportional to the number of known queries in the historical workload.

Performance of T-ASU vs. ASU: We compare the performance between ASU and T-ASU with a 95% uniformity confidence threshold on the Wiki dataset (24-1). Fig. 13 shows the performance of ASU and T-ASU. In Fig. 13(a), the Y-axis represents the sum time cost of updating both the uniform sample and the stratified sample. As a result of fewer re-samplings and smaller sample size, the time cost of T-ASU is about 70% of ASU (T-ASU: 55.7s vs. ASU: 80s). For T-ASU, re-sampling just occurs only once, because the value of $\hat{m}'$ is much larger than the new batch size so that we must re-sample from scratch to update the samples. In Fig. 13(b), due to the degeneration of randomness, the relative error of the four example queries in T-ASU has a little increment. The relative error of Q_2 in T-ASU does not increase because all tuples containing 'www' (rare subpopulation) have been extracted into the stratified sample. Consequently, the result of Q_2 will not be affected by the randomness of sampling. Moreover, since the small subset is more susceptible to the reduction of randomness than the large subset, the increased relative error of Q_1 (0.33%) and Q_4 (0.21%) is larger than that of Q_3 (0.07%). In summary, T-ASU can update the samples faster than ASU by sacrificing a little query accuracy.

Effect of the ratio of batch data over base data: We



(a) Time cost (b) Query accuracy of Q_1

Fig. 14: The performance of FULL, ASU, T-ASU, and 1% FULL on the Wiki

evaluate the effect of the ratio of batch data over base data on the Wiki dataset and compare the performance of FULL, ASU, T-ASU, and 1% FULL (re-sampling every 1% new data). We use 100 million tuples as the base data. The ratio of batch data over base data is set from 0.0001% to 10% and let the number of batches to be 100. The initial samples are constructed by full re-sampling on the base data, and the relative error of Q_1 on the initial sample is about 2%.

Fig. 14(a) shows the average sample update cost of FULL, ASU, T-ASU, and 1% FULL. The time costs of ASU and T-ASU increase little along with the increase of the ratio of batch data over base data because ASU and T-ASU need only a few re-samplings. However, the time cost of FULL and 1% FULL increases significantly as the ratio increases, especially when the ratio is high (e.g., on 10%). Since the newly ingested data generated in 100 batches on the ratio 0.0001% or 0.001% is less than 1% of the base data, the sample update will not be triggered for 1% FULL. Hence, the time cost of 1% FULL is N/A on the ratio 0.0001% and 0.001% as shown in Fig. 14(a).

Fig. 14(b) shows the relative error of FULL, ASU, T-ASU and 1% FULL for the example query Q_1. On all of the ratios, ASU and T-ASU can achieve almost equivalent query accuracy as FULL, and all of them have a smaller relative error because the updated samples guarantee the query accuracy. On the ratio of 1% and 10%, the relative error of T-ASU is larger than that of ASU, because the larger batch size will make T-ASU work under this situation. On the ratio of 0.0001%, 0.001%, and 0.01%, since the newly ingested data has not reached 1% of base data, it causes that samples cannot be maintained timely. Consequently, the relative error of 1% FULL is much larger than that of other approaches due to the stale samples. On the ratio of 0.1%, since the samples are updated for ten times, the relative error of 1% FULL becomes small. After all, because the samples are not updated timely, the relative error of 1% FULL is still larger than that of FULL. Specifically, on the ratio of 0.0001%, 0.001%, 0.01%, and 0.1%, the relative error of 1% FULL is increased on the basis of 2% until it performs a full re-sampling on the underlying dataset. On the ratio of 1% and 10%, 1% FULL is equivalent to FULL. Therefore, they have the same relative error.

V. Related Work

Sampling Approaches. Reservoir sampling [28] can construct and maintain a fixed size random sample on an unknown

size dataset. [8]–[10] and [32] focus on extending the reservoir sampling to make the sample size variable. [33] can incrementally maintain Bernoulli samples over evolving multisets. In this paper, however, we focus on how to maintain the samples with an expected sample size which satisfies the user-specified error bound.

Online Aggregation and AQP. By using the sampler operator that picks input rows uniformly-at-random with the given probability [11], [14], online aggregation methods [12], [17], [18], [20], [21], [25], [29] uniformly sample the inputs in a progressive manner that iteratively refines the approximate answers. However, the cost of online sampling will increase query latency. To achieve a much sharper reduction on response time, AQP systems draw samples from the underlying data in a pre-processing step and use them to process incoming queries [3], [6], [26], [27]. Existing works can handle the complex queries which involve joins [26] and grouping [2], [3]. For example, to address the problem that joining (uniform) samples leads to significantly fewer tuples in the output, [26] uses at most one sampled relation per join, or require the join key to be included in the stratified sample. In this paper, we focus on sample maintenance. We redesign and implement sampling engine using the approaches proposed in this paper. As described in Section II-B, we can integrate our sampling engine into the AQP system. The upper-level operations in the AQP system do not need to be changed.

VI. Conclusion

In this paper, we proposed an adaptive sample update (ASU) approach that can decide whether re-sampling is needed when a new batch of data arrives. According to this decision, ASU can make a choice between full re-sampling and incremental sample update strategy to reduce the overall sample maintenance cost. Moreover, we proposed an enhanced approach (T-ASU) to speed up sample updates by sacrificing a little query accuracy. We integrated and evaluated ASU and T-ASU with a state-of-the-art AQP engine to demonstrate their effectiveness. Extensive experiments on real-world and synthetic datasets showed that these approaches are about 1.5x-9.4x faster than the full re-sampling strategy in sample update speed while achieving almost the same query accuracy as full re-sampling does.

For future work directions, perhaps it is still to find further ways to reduce or eliminate the expensive re-sampling. As an example, we may try a predictive re-sampling method that makes use of a prediction of data distribution changes to prepare for sample size increases.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments. This work was partly supported by the National Key R&D Program of China (No. 2018YFB1004404 and 2018YFB1402600), the NSFC (No. 61732004 and No. 61802066) and the Shanghai Sailing Program (No. 18YF1401300).

References

[1] https://dumps.wikimedia.org/other/pagecounts-raw/.
[2] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. In *SIGMOD* 2002.
[3] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The Aqua Approximate Query Answering System. In *SIGMOD* 1999.
[4] K. Li and G. Li. Approximate query processing: What is new and where to go? - A survey on approximate query processing. *Data Science and Engineering*, 3(4):379–397, 2018.
[5] G. Song, W. Qu, X. Liu, and X. Wang. Approximate Calculation of Window Aggregate Functions via Global Random Sample. *Data Science and Engineering*, 3(1):40–51, 2018.
[6] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *EuroSys* 2013.
[7] Z. Wu, Y. Jing, Z. He, C. Guo, and X. S. Wang. POLYTOPE: a flexible sampling system for answering exploratory queries. *WWWJ*, 2019.
[8] M. Al-Kateb and B. S. Lee. Stratified reservoir sampling over heterogeneous data streams. In *SSDBM* 2010.
[9] M. Al-Kateb, B. S. Lee, and X. S. Wang. Adaptive-size reservoir sampling over data streams. In *SSDBM* 2007.
[10] M. Al-Kateb, B. S. Lee, and X. S. Wang. Reservoir sampling over memory-limited stream joins. In *SSDBM* 2007.
[11] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia. Spark SQL: relational data processing in spark. In *SIGMOD* 2015.
[12] B. Chandramouli, J. Goldstein, and A. Quamar. Scalable progressive analytics on big data in the cloud. *PVLDB*, 6(14):1726–1737, 2013.
[13] K. Collier. Agile Analytics: A Value-Driven Approach to Business Intelligence and Data Warehousing. Agile Software Development Series. Pearson Education, 2011.
[14] G. Cormode, M. N. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1-3):1–294, 2012.
[15] T. Finch. Incremental calculation of weighted mean and variance. University of Cambridge Computing Service, Cambridge, 2009.
[16] A. Galakatos, A. Crotty, E. Zgraggen, C. Binnig, and T. Kraska. Revisiting reuse for approximate query processing. *PVLDB*, 10(10):1142–1153, 2017.
[17] J. M. Hellerstein, R. Avnur, and V. Raman. Informix under CONTROL: online query processing. *Data Min. Knowl. Discov.*, 4(4):281–314, 2000.
[18] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *SIGMOD* 1997.
[19] B. Hentschel, P. J. Haas, and Y. Tian. Temporally-biased sampling for online model management. In *EDBT* 2018.
[20] C. M. Jermaine, S. Arumugam, A. Pol, and A. Dobra. Scalable approximate query processing with the DBO engine. In *SIGMOD* 2007.
[21] F. Li, B. Wu, K. Yi, and Z. Zhao. Wander join: Online aggregation via random walks. In *SIGMOD* 2016.
[22] S. Lohr. *Sampling : design and analysis*. Brooks/Cole, 2010.
[23] A. I. McLeod and D. R. Bellhouse. A convenient algorithm for drawing a simple random sample. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 32(2):182–184, 1983.
[24] B. Mozafari. Approximate query engines: Commercial challenges and research opportunities. In *SIGMOD* 2017.
[25] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie. Online aggregation for large MapReduce jobs. *PVLDB* 4(11):1135–1145, 2011.
[26] Y. Park, B. Mozafari, J. Sorenson, and J. Wang. Verdictdb: Universalizing approximate query processing. In *SIGMOD* 2018.
[27] L. Sidirourgos, M. L. Kersten, and P. A. Boncz. Sciborq: Scientific data management with bounds on runtime and quality. In *CIDR* 2011.
[28] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.
[29] K. Zeng, S. Agarwal, A. Dave, M. Armbrust, and I. Stoica. G-OLA: generalized on-line aggregation for interactive analysis on big data. In *SIGMOD* 2015.
[30] G. Cormode, V. Shkapenyuk, D. Srivastava, and B. Xu. Forward Decay: A Practical Time Decay Model for Streaming Systems. In *ICDE* 2009.
[31] Y. Ahmad, O. Kennedy, C. Koch, and M. Nikolic. DBToaster: Higher-order Delta Processing for Dynamic, Frequently Fresh Views. *PVLDB*, 5(10):968–979, 2012.
[32] R. Gemulla, W. Lehner, and P. J. Haas. A Dip in the Reservoir: Maintaining Sample Synopses of Evolving Datasets. In *VLDB* 2006.
[33] R. Gemulla, W. Lehner, and P. J. Haas. Maintaining Bernoulli Samples over Evolving Multisets. In *PODS* 2007.