This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2877362, IEEE Transactions on Knowledge and Data Engineering

Bounded Approximate Query Processing

Kaiyu Li Yong Zhang Guoliang Li Wenbo Tao Ying Yan

Abstract—OLAP is a core functionality in database systems and the performance is crucial to enable on-time decisions. However, OLAP queries are rather time consuming, especially on large datasets, and traditional exact solutions usually cannot meet the high-performance requirement. Recently, approximate query processing (AQP) has been proposed to enable approximate OLAP. However, existing AQP methods have some limitations. First, they may involve unacceptable errors on skewed data (e.g., long-tail distribution). Second, they require to store large amount of data and have no significant performance improvement. Third, they only support a small subset of SQL aggregation queries. To overcome these limitations, we propose a bounded approximate query processing framework BAQ. Given a predefined error bound and a set of queries, BAQ judiciously selects high-quality samples from the data to generate a unified synopsis offline, and then uses the synopsis to answer online queries. Compared with existing methods, BAQ has the following salient features. (1) BAQ does not need to generate a synopsis for each query while it only generates a unified synopsis, and thus BAQ has much smaller synopsis. (2) BAQ achieves much smaller error than existing studies. Specifically, BAQ can provide deterministic approximate results (i.e., the estimated query results must be within the error bound with 100% confidence) for SQL aggregation queries that do not contain selection conditions on numerical columns. For queries with selection conditions on numerical columns, we propose effective grouping-based techniques and the estimated results are also within the error bound in practice. Experimental results on both real and synthetic datasets show that BAQ significantly outperforms state-of-the-art approaches. For example, on a Microsoft production dataset (a real dataset with synthetic queries), BAQ has 10-100× improvement on synopsis size and 10-100× improvement on the error compared with state-of-the-art algorithms.

Index Terms—Data Integration, Approximate Query Processing, Synopsis, Sampling, Bounded Error

1 Introduction

Online analytical processing (OLAP) is an important functionality in database systems. The performance of processing OLAP queries is crucial in many applications, such as decision-support systems. For example, consider a revenue table $\mathcal{T}$ of a fruit company in Table 1. A data analyst wants to know the average tax in market "US" on category "apple", and poses a SQL query "SELECT AVG(Tax) FROM $\mathcal{T}$ WHERE Market=US and Fruit=apple". Nevertheless, executing a SQL query on large datasets is expensive and traditional exact solutions cannot meet the high-performance requirement. To address this problem, approximate query processing (AQP) [10], [18] has been proposed to enable interactive OLAP on big data. However, existing AQP techniques (e.g., sampling-based method (SAQ) [10], [12] deterministic method (DAQ) [33], Sketch [11], [13], [43], Histogram [9], [19], [26], [30]–[32], [37], [40] and Wavelet [7], [14], [15], [25], [39]) have the following limitations.

Firstly, SAQ requires a given query workload, selects a synopsis of samples for each distinct column set (QCS) of queries in the workload, and uses the synopsis to answer an OLAP query whose QCS is a subset of a given query's QCS. It is challenging to select high-quality samples. Random sampling works well on data with normal distribution, but has large (unacceptable) errors on skewed data [42]. Although stratified sampling (AQUA [35], START [36], BlinkDB [4]) can deal with sparse data, SAQ still has some limitations. (1) SAQ cannot give 100% confidence on the error bound of using samples to answer queries and may return large errors for skewed data, and thus SAQ cannot meet the requirement of many real applications that the error must be within 5% [42]. (2) SAQ requires to generate a sample for each QCS and it may generate many samples, leading to low performance.

Secondly, DAQ does not require a query workload. It focuses on numerical data and uses the high-order bits of a numerical data to approximate the results. It still needs to consider all the data and the performance improvement is limited. Moreover, it cannot support some SQL queries, e.g., queries with both categorical and numerical columns.

Thirdly, Sketch, Histogram and Wavelet also require a query workload. However, they cannot answer the queries that are not in the workload. Moreover, they can only support a subset of SQL aggregation queries and cannot support queries with categorical columns.

To address these limitations, we propose a bounded approximate query processing framework BAQ. Given an error bound and a query workload, BAQ generates a unified synopsis and uses the synopsis to answer an online query whose QCS is a subset of a given query's QCS. Compared with existing methods, BAQ has the following salient features. (1) BAQ does not need to generate a synopsis for each query while it only generates a unified synopsis, and thus BAQ generates much smaller synopsis. (2) BAQ achieves much smaller error than existing studies. Specifically, BAQ can provide deterministic approximate results (i.e., the estimated query results must be within the error bound with 100%

- *Kaiyu Li is with the Department of Computing Science, Tsinghua University, Beijing, China. liky15@mails.tsinghua.edu.cn.*
- *Yong Zhang is with the Research Institute of Information Technology at Tsinghua University, Beijing, China. zhangyong05@tsinghua.edu.cn.*
- *Guoliang Li is with the Department of Computer Science, Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University, Beijing, China. liguoliang@tsinghua.edu.cn.*
- *Wenbo Tao is with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of technology, Boston, U.S. wenbo@mit.edu.*
- *Ying Yan is with the Cloud Computing Group, MSRA, Beijing, China. ying.yan@microsoft.com.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2877362, IEEE Transactions on Knowledge and Data Engineering

confidence) for SQL aggregation queries that do not contain selection conditions on numerical columns. For queries with selection conditions on numerical columns, we propose effective grouping-based techniques and the estimated results are also within the error bound in practice.

The basic idea of BAQ is that we first judiciously select a query synopsis for each QCS with 100% confidence within the error bound, and then generate a unified synopsis by covering all these query synopses, which is used to answer online queries. For example, in Table 1, we first generate a synopsis for each query in Figure 1 and finally merge them to generate a unified synopsis in Table 2. (We will introduce more details in Section 3). To summarize, we make the following contributions.

(1) We propose a bounded approximate query processing framework BAQ. BAQ can provide deterministic approximate results for queries that do not contain selection conditions on the numerical attributes. For queries with selection conditions on numerical columns, we propose effective grouping-based techniques and the estimated results are also within the error bound in practice.

(2) We first select a query synopsis for each QCS (see Section 3). We then generate a unified synopsis by covering the query synopses, prove that the problem is NP-hard, and propose heuristic solutions (see Section 4).

(3) We extend our method to support big data and devise efficient algorithms to generate a unified synopsis on distributed environments (see Section 5).

(4) We implemented our method on Spark and evaluated our method on both real and synthetic datasets. Experiment results showed that our method generated much smaller synopsis and had much better error-bound guarantee compared with state-of-the-art algorithms (see Section 6).

2 PRELIMINARIES

2.1 Problem Definition

Data Model. Given a database relation $\mathcal{T}$ with N records, we use $\mathbb{A}$ to denote its column set, where each column $\mathcal{A} \in \mathbb{A}$ is either a *categorical* column with a limited number of distinct values or a *numerical* column with real numbers.

Query Model. Our system can support any SQL aggregation query, and here we focus on the following query.

$$\texttt{SELECT } \mathcal{A}_1^s, \mathcal{A}_2^s, \cdots, \mathcal{A}_{|\mathbb{A}^s|}^s, \texttt{AGG}(\mathcal{A}_1^a), \texttt{AGG}(\mathcal{A}_2^a), \cdots, \texttt{AGG}(\mathcal{A}_{|\mathbb{A}^a|}^a)$$
$$\texttt{FROM } \mathcal{T}$$
$$\texttt{WHERE } (\mathcal{A}_1^w \texttt{ VP } v_1) \texttt{ OP } (\mathcal{A}_2^w \texttt{ VP } v_2) \cdots \texttt{OP} \cdots (\mathcal{A}_{|\mathbb{A}^w|}^w \texttt{ VP } v_{|\mathbb{A}^w|})$$
$$\texttt{GROUP BY } \mathcal{A}_1^g, \mathcal{A}_2^g, \cdots, \mathcal{A}_{|\mathbb{A}^g|}^g$$

where $\mathbb{A}^s = \{\mathcal{A}_1^s, \mathcal{A}_2^s, \cdots \mathcal{A}_{|\mathbb{A}^s|}^s\}$ denotes a set of selected columns, $\mathbb{A}^a = \{\mathcal{A}_1^a, \mathcal{A}_2^a, \cdots, \mathcal{A}_{|\mathbb{A}^a|}^a\}$ denotes a set of aggregation columns (AGG can be COUNT, SUM, AVG, MAX, MIN), $\mathbb{A}^w$ is a set of selection conditions (VP denotes a value comparison operation in $\{<, \leqslant, >, \geqslant, =, \neq\}$, v_i denotes a value, and OP is either OR or AND), and $\mathbb{A}^g = \{\mathcal{A}_1^g, \mathcal{A}_2^g, \cdots, \mathcal{A}_{|\mathbb{A}^g|}^g\}$ is a set of grouping columns.

Given a relation $\mathcal{T}$, a set of SQL queries $q_1, q_2, \cdots, q_\tau$, and a system-defined relative error bound δ, we aim to build a synopsis $\mathcal{S}$, which is a subtable of $\mathcal{T}$ with selected records, such that we can use the synopsis to answer any query q_i and the relative error is not larger than the given bound δ.

Definition 1 (Relative Error). Given a SQL query with aggregations $\texttt{AGG}(\mathcal{A}_1^a)$, $\texttt{AGG}(\mathcal{A}_2^a)$, $\cdots$, $\texttt{AGG}(\mathcal{A}_{|\mathbb{A}^a|}^a)$, the relative error on an aggregation $\texttt{AGG}(\mathcal{A}_i^a)$ is

$$\begin{cases} \frac{|result - estimation|}{\epsilon} & if \text{ result=0} \\ \frac{|result - estimation|}{|result|} & if \text{ result}\neq 0 \end{cases} \quad (1)$$

where $result$ is the true result on $\texttt{AGG}(\mathcal{A}_i^a)$, $estimation$ is the result estimated using the synopsis, and ϵ is a small number close to 0.

Definition 2 (Bounded Synopsis Construction). Given a relation $\mathcal{T}$, SQL queries $q_1, q_2, \cdots, q_\tau$, and a system-defined relative error bound δ, select a minimal subset $\mathcal{S}$ of records based on which we compute an approximation answer for q_i such that the relative error is within δ.

Table 1 shows a table with three categorical attributes (Market, Fruit and Profitable) and two numerical attributes (Revenue and Tax). Figure 1 illustrates five example queries. Suppose the error bound is 0.2. The highlighted records in Table 1 are selected to construct the synopsis in Table 2, and we can utilize the synopsis to answer these queries within the error bound.

Result Model. We compute the deterministic approximate result where the relative error is within an error bound δ with 100% confidence.

Definition 3 (Query Column Set). Given a query q_i, the query column set (QCS) of q_i is $\pi_i = \mathbb{A}^s \cup \mathbb{A}^a \cup \mathbb{A}^w \cup \mathbb{A}^g$. $\pi = \cup_{i=1}^{\tau} \pi_i$ is the QCS of all queries.

Note that the synopsis we generate not only can answer the queries $q_1, q_2, \cdots, q_\tau$, but also can answer a query whose QCS is a subset of π_i within the error bound.

Remark. (1) Our techniques can be extended to support the nest queries and the queries with the Having clause. We omit the details due to space constraints. (2) We focus on a single table. Existing techniques for queries with multiple tables [20], [24] can be easily integrated into our method.

2.2 Workflow

Our method works in two steps: offline synopsis generation and online query processing. Figure 2 shows the architecture of our framework. The data can be stored in HDFS, HBase, and RDBMS. We generate a unified synopsis $\mathcal{S}$, store it in memory and use it to answer online queries.

Offline Synopsis Generation. The synopsis generation component generates the synopsis offline. For ease of presentation, we first discuss how to generate the synopsis for each query (Section 3) and then combine them to generate a unified synopsis (Section 4). We devise effective algorithms to generate the unified synopsis (Section 4). To support large datasets, we develop effective algorithms on Spark to generate the unified synopsis (Section 5).

Online Query Processing. Given a query q whose QCS is a subset of π_i, the query processing engine rewrites query q to q' and poses the query q' to synopsis $\mathcal{S}$ to compute the results. We will discuss how to rewrite the queries in Section 3. As our synopsis $\mathcal{S}$ is much smaller than the original table $\mathcal{T}$, the online query processing is very fast.

2.3 Related Work

Existing studies on approximate query processing can be broadly classified into two categories: AQP with known query workload and AQP without known query workload. Besides, we discuss the error estimation of AQP.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2877362, IEEE Transactions on Knowledge and Data Engineering

Query q_1
(1) `SELECT COUNT(*) FROM` $\mathcal{T}$ `GROUP BY Market`
➡ *Rewrite* q'_1: `SELECT SUM(SF) FROM` s_1 `GROUP BY Market`

Query q_2
(2) `SELECT COUNT(*) FROM` $\mathcal{T}$ `WHERE Market = US and Profitable = Yes`
➡ *Rewrite* q'_2: `SELECT SUM(SF) FROM` s_2 `WHERE Market = US and Profitable = Yes`

Query q_3
(3) `SELECT AVG(Tax) FROM` $\mathcal{T}$
➡ *Rewrite* q'_3: `SELECT` $\frac{\texttt{SUM(SF * Tax)}}{\texttt{SUM(SF)}}$ `FROM` s_3

Query q_4
(4) `SELECT MAX(Revenue) FROM` $\mathcal{T}$
➡ *Rewrite* q'_4: `SELECT MAX(Revenue) FROM` s_4

Query q_5
(5) `SELECT AVG(Tax) FROM` $\mathcal{T}$ `WHERE Market = US and Fruit = apple`
➡ *Rewrite* q'_5: `SELECT` $\frac{\texttt{SUM(SF * Tax)}}{\texttt{SUM(SF)}}$ `FROM` s_5 `WHERE Market = US and Fruit = apple`

s_1

	Market	SF
[1]	US	8
[2]	CN	2
[3]	FR	2

s_2

	Market	Profitable	SF
[1]	US	Yes	4
[2]	US	No	4
[3]	CN	Yes	2
[4]	FR	Yes	2

s_3

	Tax	SF
[1]	133	5
[2]	152	5
[3]	175	2

s_3^+

	min	max	SF
[1]	120	144	5
[2]	145	161	5
[3]	175	180	2

s_4

	Revenue	SF
[1]	1250	4
[2]	1420	5
[3]	1670	2
[4]	2130	1

s_4^+

	min	max	SF
[1]	1100	1310	4
[2]	1380	1580	5
[3]	1670	1820	2
[4]	2130	2130	1

s_5

	Market	Fruit	Tax	SF
[1]	US	orange	120	2
[2]	US	apple	150	4
[3]	US	apple	161	2
[4]	CN	banana	152	2
[5]	FR	banana	144	2

Fig. 1. Five Example Queries and Their Synopses.

	Market	Fruit	Profitable	Revenue	Tax
r_1	**US**	**orange**	**Yes**	**1100**	**120**
r_2	US	orange	No	1250	140
r_3	US	apple	Yes	1400	145
r_4	**US**	**apple**	**No**	**1380**	**150**
r_5	**CN**	**banana**	**Yes**	**1420**	**152**
r_6	**US**	**apple**	**Yes**	**1670**	**161**
r_7	**FR**	**banana**	**Yes**	**1820**	**144**
r_8	US	apple	No	1500	175
r_9	FR	banana	Yes	1310	125
r_{10}	US	apple	Yes	1580	133
r_{11}	CN	banana	Yes	1220	180
r_{12}	**US**	**apple**	**No**	**2130**	**154**

TABLE 1
Table $\mathcal{T}$.

Market	Fruit	Profitable	Revenue	Tax	SF_2	SF_4	SF_5
US	orange	Yes	1100	120	4	4	2
US	apple	No	1380	150	4	5	4
CN	banana	Yes	1420	152	2	0	2
US	apple	Yes	1670	161	0	2	2
FR	banana	Yes	1820	144	2	0	2
US	apple	No	2130	154	0	1	0

TABLE 2
A Unified Synopsis for Five Queries.

2.3.1 AQP With Known Query Workload

Given a query workload and a predefined error bound, existing algorithms aim to generate synopsis offline and use the synopsis to answer online queries. There are two cases for the known query workload: query matching and QCS matching. The former answers online queries which exactly match a query in the query workload. The latter answers an online query if there exists a query in the workload whose QCS contains the online query's QCS.

(1) Query Matching: Wavelet, Sketch and Histogram use query matching. For each query, Histogram groups the numerical values into 'buckets'. For each bucket, Histogram computes its summary statistics that can be used to approximately answer a query. Wavelet is conceptually close to Histogram, and it aims to compress the data by capturing the most expressive features. However, Wavelet requires to decompress when answering queries. Sketch models a numerical column as a vector or matrix and transforms the data by a fixed matrix (depend on query workload) to construct the synopsis. Sketch is suit for streaming data but not for general relational database.

These methods have several limitations. First, they need to know all the queries in advance. Second, they only focus on numerical columns. Third, they take much space to store the synopses, as they construct a synopsis for each query.

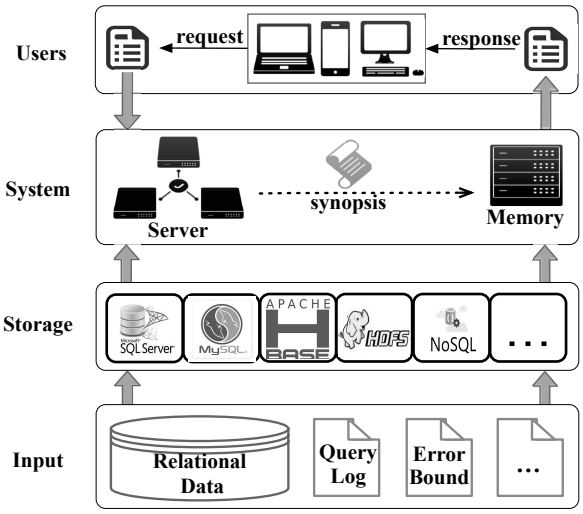


Fig. 2. Architecture of BAQ.

(2) QCS Matching: Sampling-based approximate query processing (SAQ) is widely used to support QCS matching. SAQ first selects samples for each QCS and then uses the samples to answer online queries [10], [27], [41]. It is challenging to select high-quality samples and many methods are proposed to generate high-quality samples. A common method is random sampling [42]. However, this method can only provide a high confidence (e.g., 95%) that the result of using the sample to answer a query is within a given error bound, but it cannot achieve 100% confidence. Moreover, it performs badly in skewed data, e.g., long-tail distribution. Besides, the size of groups and the values in each group may be highly skewed, making many traditional uniform-sampling-based methods unreliable [42]. Although stratified sampling (AQUA [35], START [36], BlinkDB [4], Babcock [5], Sample + seek [12]) can deal with sparse data, SAQ still has some limitations. (i) They generate samples for each QCS and cannot share the samples among different QCSs. If we simply union the independent samples for every attribute, the overall sampling rate will be prohibitively large. Instead, our method BAQ shares samples among QCSs and the synopsis size is rather small. (ii) SAQ ignores the long tails of data

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2877362, IEEE Transactions on Knowledge and Data Engineering

distributions, making it hard to effectively answer MAX/MIN queries while our method can support MAX/MIN very well. (iii) To provide high confidence, SAQ uses Bootstrap [3], [17], [29], [44] to diagnose the results by using multiple samples. This approach can be computationally expensive and inaccurate for some queries.

2.3.2 AQP Without Query Workload

(1) Deterministic Approximate Query: Deterministic approximate queries (DAQ) [33] used the high-order bits of numerical data to do approximation, by borrowing the idea from probabilistic databases [22], [23], [38]. However, DAQ cannot support general SQL aggregation queries with Where and Groupby while BAQ can. Moreover, DAQ maintains all records and the performance improvement is limited.

(2) Online Aggregation: Online aggregation (OLA) provides the users with approximate answer with online estimation based on currently seen data [8], [16], [24], [28], [34]. The user can stop the query execution if she is satisfied with the answer. OLA has no guarantee on the performance and error bound, and it takes long time and returns bad results.

2.3.3 Error Estimation

(1)Error Estimation with Known Distribution. If we have a-priori knowledge about the data distribution or have enough samples to get the distribution, then we can regard it as a parameter estimation problem. For real-world datasets, many datasets follow the normal distribution and existing studies assume that the data follows normal distribution [6].

(2)Error Estimation without Known Distribution. If the distribution is unknown, some resampling methods, e.g., Bootstrap, can be used. The key idea of Bootstrap is that to use sample S to replace original dataset D, one can draw samples from S instead of D to compose the distribution of AGG(S) [44]. For each sample S, the estimated values of AGG(S) can be computed and these values compose a distribution which can be used to estimate the aggregation result. Interested readers are referred to [17] for more details. Closed-form estimation is faster than resampling in some specific queries. In probability theory, the central limit theorem (CLT) establishes that, for independent random variables, the normalized sum tends toward a normal distribution (informally a "bell curve") even if the original variables themselves are not normally distributed. Thus, the distribution of AGG (S) can be approximated as N(AGG (S),σ^2), where σ can be computed by the mean squared error Var(S). Computing Var(S) for a small dataset S will be faster than the brute-force resampling. However, this method can only work for COUNT, SUM, AVG but fail to deal with the queries whose variance is hard to compute such as $\max$, $\min$ or user-defined functions [3]. Large deviation bounds studies how to compute the confidence interval in the worst case by estimating the minimal and maximal values meanwhile keeping the results not dominated by the outliers.

2.4 Comparison with Existing Methods

We compare our method BAQ with existing methods in Table 3. (1) **Supported Queries.** Wavelet, Histogram and Sketch assume the queries are known while BAQ assumes QCSs are known. In other words, Wavelet, Histogram and Sketch can only answer queries that appear in the query workload, while BAQ and SAQ can answer queries whose QCSs are contained by the QCS of a query in the workload. Wavelet, Histogram and Sketch only support numerical columns while BAQ can support both categorical and numerical columns. (2) **Synopsis Size.** BAQ generates a smaller unified synopsis while SAQ generates larger samples. DAQ provides approximation using high-order bits of attributes thus needs to store and access all of the tuples while BAQ just stores a unified synopsis. Thus BAQ has the smallest synopsis size. (3) **Online Efficiency.** As BAQ has the smallest synopsis size, BAQ achieves the highest online efficiency. (4) **Error Bound.** BAQ provides exact answer for categorical only queries and deterministic approximation results for queries without selection conditions on numerical columns while other methods cannot. For queries with selection conditions on numerical columns, none of these methods can provide deterministic approximation results while BAQ can provide results within error bounds in practice as verified in our experiments. Thus, BAQ can provide answers with smaller errors. SAQ performs bad on skewed data especially MAX and MIN, while BAQ is not sensitive to the data distribution.

	Workloads	Skewed Data	Error Bound Guarantee		
			Categorical Only	No Numerical in WHERE	Numerical in WHERE
SAQ	QCS	×	not 100%	not 100%	not 100%
DAQ	No	√	not 100%	not 100%	not 100%
Wavelet	Queries	√	×	not 100%	not 100%
Histogram	Queries	√	×	not 100%	not 100%
Sketch	Queries	√	×	not 100%	not 100%
BAQ	QCS	√	Exact	100%	not 100%

TABLE 3
Comparison of BAQ with state-of-the-arts. (×: Not support; √: Support such case; Exact: Exact result; 100%: 100% within δ.)

3 Single Query

We study how to build a synopsis for a single query. Given a query q_i and its QCS π_i, we first study the case that π_i only contains categorical columns in Section 3.1. Then we discuss the case that π_i contains both categorical and numerical columns but the where clause does not contain the numerical columns in Section 3.2. Next we consider that the where clause contains numerical columns in Section 3.3.

3.1 Categorical Columns Only

Synopsis Definition. For simplicity, we first consider a simple case that the QCS of q_i contains only one column, i.e., $|\pi_i| = 1$. In this case, suppose the attribute is $\mathcal{A}$. Let $\mathcal{D}[\mathcal{A}]$ denote the set of *distinct* values in column $\mathcal{A}$. For each value $v \in \mathcal{D}[\mathcal{A}]$, we compute its frequency in table $\mathcal{T}$, denoted by $f[v]$. We call $f[v]$ the *scale factor* (SF) of v. We select $(v, f[v])$ for every $v \in \mathcal{D}[\mathcal{A}]$ as synopsis. For example, consider query q_1 in Figure 1. There are 3 distinct values and the scale factor of "US" is 8. Then we consider that $|\pi_i| > 1$. Let $\mathcal{T}[\pi_i]$ denote the subtable of $\mathcal{T}$ that only keeps the columns in π_i, and $\mathcal{D}[\pi_i]$ denote the set of distinct tuples in $\mathcal{T}[\pi_i]$. For each tuple $V \in \mathcal{D}[\pi_i]$, we compute the frequency of tuple V in $\mathcal{D}[\pi_i]$, denoted by $f[V]$. We call $f[V]$ the *scale factor* of V. We select $(V, f[V])$ for every $V \in \mathcal{D}[\pi_i]$ as synopsis. For example, consider the SQL query q_2 in the Figure 1. There are 4 distinct tuples of (Market, Profitable) and the scale factor of "(US, Yes)" is 4.

Definition 4. (SYNOPSIS OF q_i WITH CATEGORICAL COLUMNS ONLY) The synopsis of q_i is a table with $|\pi_i|+1$ columns, including $|\pi_i|$ columns in π_i and a column SF. Each row contains tuple $V \in \mathcal{D}[\pi_i]$ and its frequency $f[V]$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2877362, IEEE Transactions on Knowledge and Data Engineering

For example, q_1 and q_2 in Figure 1 only contain categorical columns and the synopses s_1, s_2 are shown in the figure.

Synopsis Construction. We can easily generate synopsis by scanning the records in $\mathcal{T}$ once. We scan the records in $\mathcal{T}$ and keep a hash table to keep the tuples in $\mathcal{T}[\pi_i]$. If a tuple in $\mathcal{T}[\pi_i]$ appears in the hash table, we increase the frequency by 1; otherwise we add the tuple into the hash table and set the frequency as 1. After accessing all records in $\mathcal{T}$, we generate the synopsis. The time complexity is $\mathcal{O}(N)$.

Query Rewriting. If q_i contains categorical columns only, `AGG` can only be `COUNT`, because it does not make sense for `SUM`, `MIN`, `AVG`, `MAX` on categorical columns. For any column $\mathcal{A}$, we have `COUNT`($\mathcal{A}$)=`SUM`(`SF`). Thus, we can rewrite q_i by replacing each `COUNT`($\mathcal{A}$) with `SUM`(`SF`). For example, we show the rewritten queries in Figure 1.

Error Analysis. For a query q_i with only categorical columns, we can use the synopsis s_i to exactly answer the queries with the same `QCS` as q_i (no error) as stated in Lemma 1.

Lemma 1. For each query q_i with only categorical columns, the result of using the synopsis s_i to answer the queries with the same `QCS` as q_i is exactly the same as the result of using the original data to answer the queries.

Proof 1. Due to space constraints, we put all proofs in our technical report[1].

3.2 No Numerical Columns in The Where Clause

We first consider that π_i only contains numerical columns in Section 3.2.1 and then discuss the case that π_i contains both numerical and categorical columns in Section 3.2.2.

3.2.1 Numerical Columns Only

Obviously, the `groupby` clause does not have numerical columns, and thus in this case, the `where` clause is empty and the select clause has only numerical columns.

(1) Only One Numerical Column. Values in numerical columns are usually continuous and it is expensive and not practical to select distinct values for numerical columns. For example, considering the `Tax` column, the number of distinct values is the same as the number of records in the table. To address this issue, we propose to partition the numerical values in column $\mathcal{A}$ into a set of disjoint groups. Suppose the values in column $\mathcal{A}$ are $v_1, v_2, \cdots, v_N$. Without loss of generality, we assume that the values are sorted (if not sorted, we first sort the values), i.e., $v_j \leq v_k$ for $j < k$. Next we discuss how to generate the groups.

Definition 5 (Numerical Value Grouping). We partition the values into x groups, $G[1] = \{v_1, \cdots, v_{j_1}\}$, $G[2] = \{v_{j_1+1}, \cdots, v_{j_2}\}$, $\cdots$, $G[x] = \{v_{j_x+1}, \cdots v_N\}$ where the relative error between the smallest value and the largest value in each group is not larger than δ, i.e., $\frac{v_{j_t} - v_{j_{t-1}+1}}{v_{j_{t-1}+1}} \leq \delta$ for $1 \leq t \leq x+1$, $j_0 = 0, j_{x+1} = N$.

As the relative error between any two values in each group is not larger than δ, we can select any value as a `pivot` to represent the values, and the relative error of other non-selected values to the pivot is not larger than δ.

1. http://dbgroup.cs.tsinghua.edu.cn/ligl/baq.pdf

Grouping Strategy. We first argue that we can find a strategy to generate the groups. A trivial case is that each value forms a group and there are N groups. However, this method will involve a huge number of groups. For example, if we use each of the distinct values in `Tax` column as a group, there will be 12 groups. Thus, we aim to find a grouping strategy to minimize the number of groups.

Minimizing The Number of Groups. We propose a scan-based algorithm to minimize the number of groups. Firstly, we initialize the first group and add the first value v_1 into the group. Then, we consider the next value v_2. If $v_2 \leq v_1 * (1+\delta)$, we add v_2 into the first group; otherwise we generate a new group and put v_2 into the new group. Iteratively we can generate all the groups. For example, consider the column `Tax`. We first sort the values in `Tax`. Suppose the relative error is $\delta = 0.2$. We add the first value 120 into the first group and then insert 125, 133, 140, 144 into the first group. We find that $145 > (1+0.2) * 120 = 144$, so we generate a new group and put 145 into the second group. Iteratively we can partition the values into 3 groups.

Lemma 2. The scan-based algorithm can minimize the number of groups.

Synopsis Definition. We partition the numerical values into groups $G[1], G[2], \cdots, G[x]$. For each group $G[p](p \in [1,x])$, we select a pivot $v \in G[p]$ and take the size of $G[p]$ as the scale factor of v. We select $(v, |G[p]|)$ as the synopsis.

Synopsis Construction. Given a query with numerical column $\mathcal{A}$, we first sort the values and generate groups for the column. Then we select a pivot from each group to generate the synopsis. The complexity is $\mathcal{O}(N \log N)$.

Query Rewriting. If q_i contains numerical columns only, `AGG` can be `COUNT`, `MIN`, `MAX`, `SUM`, `AVG`. We can rewrite query q_i by replacing `AGG`($\mathcal{A}$) as below:

- `COUNT`($\mathcal{A}$) $\longrightarrow$ `SUM(SF)`
- `MAX`($\mathcal{A}$) $\longrightarrow$ `MAX`($\mathcal{A}$)
- `MIN`($\mathcal{A}$) $\longrightarrow$ `MIN`($\mathcal{A}$)
- `SUM`($\mathcal{A}$) $\longrightarrow$ `SUM`($\mathcal{A}$*`SF`)
- `AVG`($\mathcal{A}$) $\longrightarrow$ $\frac{\texttt{SUM}(\mathcal{A}*\texttt{SF})}{\texttt{SUM(SF)}}$

Error Analysis. We discuss errors for different functions.

(1) `COUNT`($\mathcal{A}$). There is no error because we can use the scale factor to exactly count the number.

(2) `MAX`($\mathcal{A}$) and `MIN`($\mathcal{A}$). The pivot and the correct answer must be in the same group. According to Definition 5, the error of computing `MAX`($\mathcal{A}$) and `MIN`($\mathcal{A}$) is within δ.

(3) `SUM`. Let v_i denote a value and v_i' is the pivot that is used to represent v_i. The relative error of `SUM` is

$$\left|\frac{\sum_{k=1}^N v_i - \sum_{k=1}^N v_i'}{\sum_{k=1}^N v_i}\right| = \left|\frac{\sum_{k=1}^N (v_i - v_i')}{\sum_{k=1}^N v_i}\right| \leq \frac{\sum_{k=1}^N |v_i - v_i'|}{\sum_{k=1}^N |v_i|} \leq \frac{\sum_{k=1}^N \delta * |v_i|}{\sum_{k=1}^N |v_i|} = \frac{\delta * \sum_{k=1}^N |v_i|}{\sum_{k=1}^N |v_i|} = \delta$$

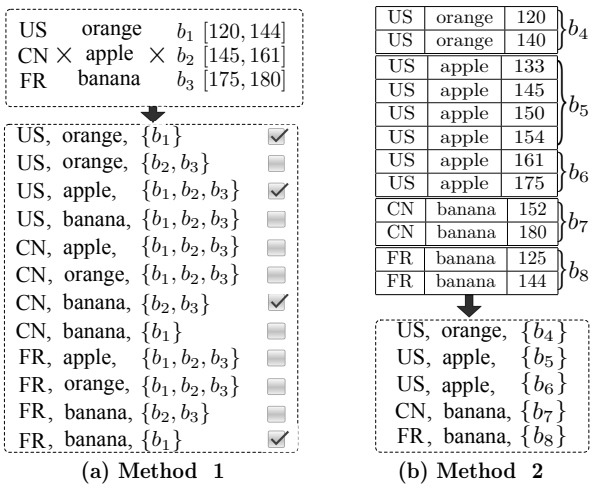
(4) `AVG`. The relative error of `AVG` is:

$$\left|\frac{(\sum_{k=1}^N v_i - \sum_{k=1}^N v_i')/N}{(\sum_{k=1}^N v_i)/N}\right| = \left|\frac{\sum_{k=1}^N v_i - \sum_{k=1}^N v_i'}{\sum_{k=1}^N v_i}\right| \leq \delta$$

Thus the relative errors for all functions are within δ

For example, consider q_3 in Figure 1. The rewritten query q_3' and synopsis s_3 are shown in the figure. We partition the numbers in `Tax` and add the scale factor. We answer q_3' by

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2877362, IEEE Transactions on Knowledge and Data Engineering



(a) Method 1 (b) Method 2

Fig. 3. Examples of Two Grouping Methods.

computing $\frac{133*5+152*5+175*2}{12} = 147.92$. The real answer is 148.25, and the relative error is $\frac{|147.92-148.25|}{148.25} = 0.00223$.

(2) Multiple Numerical Columns. We partition each numerical column into groups and store them independently. The query rewriting is the same as the queries with a single numerical column. The relative errors are still within δ.

3.2.2 Both Numerical and Categorical Columns

For simplicity, we first introduce some notations. Let π_i^c denote the set of categorical columns in π_i and π_i^n denote the set of numerical columns in π_i. Let $\mathcal{D}[\pi_i^c]$ denote the set of distinct tuples in $\mathcal{T}[\pi_i^c]$ and $\mathcal{G}[\pi_i^n]$ denote the set of group combination for numerical columns in π_i^n.

We propose two methods to generate the synopsis. The first is a combination method that conducts a Cartesian product on $\mathcal{D}[\pi_i^c]$ and $\mathcal{G}[\pi_i^n]$ as follows. For each tuple V in $\mathcal{D}[\pi_i^c]$, for each group G in $\mathcal{G}[\pi_i^n]$, it generates a pair (V, G) and counts the number of records in $\mathcal{T}$ whose π_i^c tuple is V and whose π_i^n value is in group G, denoted by $f(V, G)$. If $f(V, G) > 0$, we add $(V, G, f(V, G))$ into the synopsis. For example, consider the SQL query q_5 in Figure 1 with two categorical columns and a numerical column. Considering the 3 columns. Suppose we group the numerical column with $\delta = 0.2$. It can be divided into 3 groups b_1, b_2 and b_3. There are 3 distinct values in each categorical column. Thus there are 27 combinations. Since there are no tuples in some of the combinations, we generate 7 tuples in the synopsis as shown in Figure 3(a).

The second method first enumerates all the distinct tuples of $\mathcal{D}[\pi_i^c]$. Then for each tuple V in $\mathcal{D}[\pi_i^c]$, we consider the subtable of $\mathcal{T}$ whose π_i^c tuple is V, i.e., $\mathcal{T}[\pi_i^c = V]$. Then we group these tuples based on the numerical values on π_i^n. We still generate the minimum number of groups. For each group G, we take the size of the groups on $\mathcal{T}[\pi_i^c = V]$ as the scale factor, denoted by $f(G, \pi_i^c = V)$. We add $(V, G, f(G, \pi_i^c = V))$ into the synopsis. For example, in Figure 3(b), if we group it according to the categorical columns, it will be divided into to (`US`,`orange`), (`US`,`apple`), (`CN`,`banana`), (`FR`,`banana`) groups. We set $\delta = 0.2$. The tuples in the group (`US`,`apple`) will be divided into b_5, b_6. This method generates 5 groups, which has less groups than the first method.

Lemma 3. With the same relative error bound, the second method generates no larger synopsis than the first.

In this case, the query rewriting is the same as the previous sections. The complexity of generating the synopsis is $\mathcal{O}(|\pi_i^n| N \log N)$, where $|\pi_i^n|$ is the number of numerical columns in the query.

3.3 The Where Clause Has Numerical Columns

In this section, we focus on the case that the `where` clause contains numerical columns, e.g., $\mathcal{A} > 100$. We still use the above methods to generate the synopsis but the way of using synopsis to answer a query is different.

3.3.1 Only One Numerical Column in `Where` Clauses

We consider the case that there is only one numerical column in the `where` clause, e.g., $\mathcal{A} > v$, and our method can be easily extended to support other cases, e.g., $\mathcal{A} < v$.

Suppose there are x groups for $\mathcal{A}$, $G[1], G[2], \cdots, G[x]$. We consider the following cases.

(1) There is no group satisfying $\mathcal{A} > v$, i.e., the maximal value of $G[x]$ is smaller than v. Then there is no tuple satisfying the query, and our method can exactly answer the query.

(2) There is only one group satisfying $\mathcal{A} > v$, i.e., the maximal value of $G[x]$ is larger than v but the minimal value of $G[x]$ is smaller than v. In this case, some tuples in the group satisfy the query, and $G[x]$ is called *a partially satisfied group*.

(3) There are more than one groups satisfying $\mathcal{A} > v$, i.e., the maximal value of $G[j]$ is larger than v and the minimal value of $G[j]$ is smaller than v. Then $G[j]$ is a partially satisfied group and $G[k > j]$ is a fully satisfied group (as all tuples in $G[k]$ satisfy the query condition).

(4) All groups satisfy $\mathcal{A} > v$, i.e., the minimal value of $G[1]$ is larger than v. Our method can exactly answer the query.

For (1) and (4), we just use the same rewritten query and synopsis to answer a query. For (2) and (3), we process the fully satisfied group and partially satisfied group separately and then sum up the results of the two cases.

Partially Satisfied Group. For the partially satisfied group (there is at most one partially satisfied group), we need to estimate how many tuples in the group satisfy the query condition. Suppose G is the partially satisfied group with a scale factor $G.f$. Let $G.min$ and $G.max$ denote the minimal and maximal values in the group. Then the number of tuples satisfying the query condition is estimated as $\lfloor G.f * \frac{G.max - v}{G.max - G.min} \rfloor$. To this end, we maintain a table s_i^+, which keeps three columns $\mathcal{A}_m$, $\mathcal{A}_M$, $\mathcal{A}_f$ to store each group's minimum/maximum value and the scale factor.

Fully Satisfied Group. For fully satisfied group, we still use the rewritten query to get the answer, and the error bound is also within δ. For example, consider the following query:

`SELECT AGG(Tax) FROM` $\mathcal{T}$ `WHERE Tax` > 134

We can also use the table s_3^+ in Figure 1 to answer this query. Groups (145, 150, 152, 154, 161), (175, 180) are fully satisfied, (120, 125, 133, 140, 144) is partially satisfied. We store the minimal number and maximal number in s_3^+ as [120,144], [145,161], [175,180] and the scale factors are 5, 5

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2877362, IEEE Transactions on Knowledge and Data Engineering

and 2 respectively. In the partially covered group, the scale factor is 5. Then the number of tuples satisfied the query is estimated as $\lfloor 5 * \frac{144-134}{144-120} \rfloor = 2$.

Next we give the details on how to utilize this technique to rewrite a query.

(1) MAX and MIN:

Query Rewritting. When AGG is MAX or MIN and the result is in a partially satisfied group, we directly use $\texttt{MAX}(\mathcal{A})$ and $\texttt{MIN}(\mathcal{A})$ to rewrite the query without any change.

- $\texttt{MAX}(\mathcal{A}) \longrightarrow \texttt{MAX}(\mathcal{A})$, $\texttt{MIN}(\mathcal{A}) \longrightarrow \texttt{MIN}(\mathcal{A})$

Error Analysis. It is obvious that the result is within δ.

(2) COUNT:

Query Rewriting. The rewritten query contains two parts. Firstly, we compute the value $G.max$ in the partially satisfied group and estimated number of tuples satisfying $\mathcal{A} > v$ in the partially satisfied group by

$$\texttt{SELECT } \mathcal{A}_M, \lfloor \texttt{SF} * \frac{\mathcal{A}_M - v}{\mathcal{A}_M - \mathcal{A}_m} \rfloor \texttt{ FROM } s^+ \texttt{ WHERE } \mathcal{A}_m \leq v \leq \mathcal{A}_M$$

Secondly, we compute the number of tuples satisfying $\mathcal{A} > v$ in fully satisfied groups by:

$$\texttt{SELECT SUM(SF) FROM } s^+ \texttt{ WHERE } \mathcal{A}_m > G.max$$

Then the sum of results from these two SQLs is the answer of the count query. In above example query, we have estimated 2 tuples in the partially satisfied group and we know that there are 5+2=7 tuples in the fully satisfied group. Then the answer is 7+2=9. It is the same to the exact answer.

Error Analysis. There is no error in the fully satisfied group. The error in the partially satisfied group is small. Fortunately, the partially satisfied group is always much smaller than the fully satisfied groups, so the answer is reliable.

(3)SUM:

Query Rewriting. The rewritten query contains two parts. We compute the maximum value $G.max$ and the SUM in the partially satisfied group by

$$\texttt{SELECT } \mathcal{A}_M, \mathcal{A}_M * \lfloor \texttt{SF} * \frac{\mathcal{A}_M - v}{\mathcal{A}_M - \mathcal{A}_m} \rfloor$$
$$\texttt{FROM } s^+ \texttt{ WHERE } \mathcal{A}_m \leq v \leq \mathcal{A}_M$$

We compute the result in the fully satisfied part as:

$$\texttt{SELECT SUM}(\frac{\mathcal{A}_\texttt{m} + \mathcal{A}_\texttt{M}}{2} * \texttt{SF}) \texttt{ FROM } \texttt{s}^+ \texttt{ WHERE } \mathcal{A}_\texttt{m} > \texttt{G.max}$$

Then we can add them up. In above example query, the summation of partially satisfied group is 144*2=288 and the summation in fully satisfied groups is $\frac{145+161}{2} * 5 + \frac{175+180}{2} * 2 = 1120$. The final result is 1120+288=1408. The exact answer is 1401.0 and the relative error is only 0.0049.

Error Analysis. The error in fully satisfied groups is within δ. The error in the partially satisfied group is small.

(4)AVG:

Query Rewriting. We can compute the answer by dividing the result in (3) by the result in (2). For example, If the aggregation function is AVG, we can just use $\frac{1408}{9} = 156.44$. The exact answer is 155.67 and the relative error is 0.0050.

Error Analysis. The AVG in the fully satisfied groups will be within error bound. The additional error is caused by the partially satisfied groups when counting the number of satisfying records, and the error is very small.

3.3.2 *Multiple Numerical Columns in* `Where` *Clauses*

When the `where` clause contains multiple numerical columns, e.g., $\mathcal{A}_1 > v_1$ and $\mathcal{A}_2 > v_2$, there may be many fully satisfied groups and one partially satisfied group for each predicate. We first identify the partially satisfied group and fully satisfied group. Then we consider the four combinations of the two columns (and the method can be easily extended to support more than 2 predicates): partially/partially, partially/fully, fully/partially, and fully/fully. For fully/fully, we use the rewritten query and the error is within the bound. For partially/fully or fully/partially, we use methods in Section 3.3.1 to estimate the results. For partially/partially, we assume the two columns are independent and estimate the answer.

Remark. Even there are additional errors caused by partially satisfied groups in where clause according to theoretical analysis, but the experimental results indicate that these errors are very small. We show the details in Section 6.

4 MULTIPLE QUERIES

In this section, we study how to build synopsis for multiple queries. A straightforward method builds a synopsis for each QCS and then uses multiple synopses to answer queries. Obviously this method generates many synopses and incurs high space and time cost. We find that some synopses are redundant, and we discuss how to detect and eliminate the redundant synopses in Section 4.1. Then we propose to combine the multiple synopses and generate a unified synopsis to answer multiple queries. We discuss how to minimize the synopsis size for multiple queries in Section 4.2. Since the problem of minimizing the synopsis size is NP-hard, we propose two approximate algorithms in Section 4.3. We discuss how to answer newly coming queries and incrementally update the synopsis in Section 4.4.

4.1 Redundant Synopsis Detection

Consider two queries q_i, q_j and their QCS π_i, π_j. If $\pi_i \subseteq \pi_j$, q_j has more attributes than q_i and generates finer granularity synopsis than q_i. We find that the synopsis s_j of q_j can be used to answer q_i as follows. We still use the same method to rewrite query q_i as q_i'. Then we pose query q_i' to synopsis s_j, and get a result $A(s_j, q_i')$. We can prove that the error bound of result $A(s_j, q_i')$ is also within δ as stated in Lemma 4, because if there is a tuple in the synopsis of q_i, then there must be one or more corresponding tuples in the synopsis of q_j.

Lemma 4. Given two queries q_i, q_j and their QCS π_i, π_j, if $\pi_i \subseteq \pi_j$, the error of using s_j to answer q_i is within δ.

For example, considering q_1 and q_2 in the Figure 1. $\pi_1 = \{\texttt{Market}\} \subseteq \{\texttt{Market}, \texttt{Profitable}\} = \pi_2$. s_2 can be used to answer q_1, and q_1 in Figure 1 is equivalent to

$$\texttt{SELECT COUNT}(*) \texttt{ FROM } \mathcal{T} \texttt{ GROUP BY Market}$$
$$\texttt{WHERE Profitable} = \texttt{Yes OR Profitable} = \texttt{No}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2877362, IEEE Transactions on Knowledge and Data Engineering

The scale factor of 'US' is 8 in s_1. It is 4+4=8 in s_2.

Lemma 5. Given a query q_i in the query workload, and online query q, if q's QCS is a subset of π_i, the error of using s_i to answer q is within δ.

Definition 6. Given a set of queries, if $\pi_i \subseteq \pi_j$, the synopsis of π_i is redundant.

We aim to eliminate all redundant synopses. To this end, we enumerate every pair of queries (q_i, q_j). If the QCS of q_i is the subset of the QCS of q_j, we do not keep the synopsis of q_i. For example, consider the five queries q_1 to q_5 in Figure 1. The QCSs are {Market, (Market, Profitable), Tax, Revenue, (Market, Fruit, Tax)}. We can eliminate Tax and Market because Market $\subseteq$ (Market, Profitable) and Tax $\subseteq$ (Market, Fruit, Tax).

4.2 Constructing A Unified Synopsis

We aim to generate a single synopsis and utilize the synopsis to answer online queries within the threshold δ.

Synopsis Definition. Without loss of generality, suppose $q_1, q_2, \cdots, q_\tau$ are the queries after removing the redundant ones. We use π to denote the set of distinct columns of these queries. The synopsis has $|\pi| + \tau$ columns, including columns in π and the scale factor for each query. For each synopsis s_i of query q_i, consider the j-th tuple $s_i[j]$ and its corresponding scale factor f_i^j in its synopsis. A k-th record $r[k]$ in original table $\mathcal{T}$ covers $s_i[j]$, if (1) $s_i[j]$ and $r[k]$ have the same value on categorical columns in q_i, and (2) $s_i[j]$ and $r[k]$ are in the same group in each numerical column. If there are multiple records covering $s_i[j]$, we assign the scale factor $\mathtt{SF}_{\pi_i}$ of one tuple as f_i^j and those of others as 0.

We use "record" to refer to data in the table $\mathcal{T}$ and "tuple" to refer to the data in a synopsis for ease of presentation.

For example, consider the 3 queries q_2, q_3, q_5 after removing the redundancy. $\pi = \pi_2 \cup \pi_3 \cup \pi_5$ =(Market, Fruit, Profitable, Revenue, Tax). We use columns in π and 3 scale factor columns for the three queries to construct the unified synopsis as shown in Table 2. Considering the first tuple $s_5[1]$ in s_5 in Figure 1, the first record r_1 in Table 2 covers $s_5[1]$ because they have the same value in categorical columns {Market, Fruit}, and in numerical column Tax, the two values (120 and 120) are in the same group.

Synopsis Construction. We want to select the minimum number of records in the unified synopsis $\mathcal{S}$ to cover all tuples in each query synopsis such that we can utilize $\mathcal{S}$ to answer every query. However, this problem is NP-hard by a reduction from the minimum cover problem [21] as stated in Lemma 6.

Lemma 6. Selecting a unified synopsis to cover all the query synopses with the minimum size is NP-Hard.

Lemma 6 implies that we need to find effective approximation algorithms. We will introduce two heuristic algorithms in the following subsection.

Query Processing with the Unified Synopsis. Given a query q, if its columns are contained by a QCS π_i, we use $\mathcal{S}$ to answer the query using the same rewritten query q_i' within the bound δ; otherwise, we use the original data to answer the query.

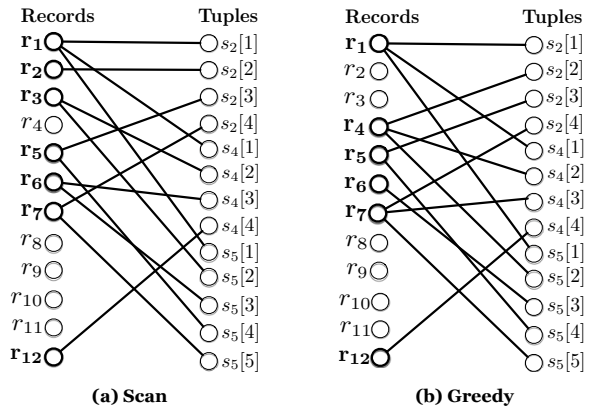


Fig. 4. Unified Synopsis Generation Algorithms.

Lemma 7. The answer of the rewritten query q_i' of q_i on the unified synopsis $\mathcal{S}$ is the same as the answer of q_i' on synopsis s_i (within error threshold δ).

For example, consider q_5 in Figure 1. The exact result of q_5 on $\mathcal{T}$ is 153.0. When we use q_5 or q_5' on the synopsis s_5, the result is 153.67 and the error is $\frac{(153.67-153.0)}{153.0} = 0.0044$.

Remark. We cannot construct a synopsis for $\pi = \cup_{i=1}^{\tau} \pi_i$ which is a superset of each π_i. In real workload, the queries may be related to many columns, and if we enumerate all the possible values of π, there will be a large number of groups and the size of synopsis will be close to the original table $\mathcal{T}$. Moreover, it is unnecessary to construct a synopsis for π because if there does not exist a query that contains two columns in π, we do not need to combine them. For example, if we use the method in Section 3.2.2 to construct a synopsis for $\pi = \cup_{i=1}^{5} \pi_i$=(Market, Fruit, Profitable, Revenue, Tax), we first partition the table into 6 categories according to the 3 categorical columns and finally get a synopsis with 9 tuples after grouping each of the category. Additionally, we should add 3 columns for scale factors. It is close to the size of the original table $\mathcal{T}$ (12 records). So we cannot use this strategy to construct a synopsis.

4.3 Synopsis Finding Algorithms

We propose approximate algorithms to generate the synopsis. We first introduce a scan-based algorithm which is very fast but has large synopsis size. We then introduce a greedy algorithm which is slower but has small synopsis size.

4.3.1 Scan-based Algorithm

We first generate the query synopsis for each query. Then we propose a scan-based method to find a number of records to cover each tuple in query synopses. We use a flag to keep the status of tuple $s_i[j]$ of synopsis s_i. Initially all of the flags are set false. We scan the table $\mathcal{T}$ and for each record r_k, we enumerate each synopsis s_i. If r_k covers a tuple $s_i[j]$ in s_i with the false flag, we add r_k into the synopsis $\mathcal{S}$ and set the flag of each $s_i[j]$ covered by r_k as true, where $1 \le j \le |s_i|$. (Note if r_k covers multiple tuples for different queries, we only add r_k into $\mathcal{S}$ once.) If all the flags are true, the algorithm terminates. Note that this algorithm must terminate because for each tuple $s_i[j]$ in a synopsis, we can select the corresponding record (the tuple is generated from) which must cover this tuple.

For example, considering the table $\mathcal{T}$ in Table 1, we set all the tuples in s_2, s_4, s_5 in Figure 1 as false. The

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2877362, IEEE Transactions on Knowledge and Data Engineering

first record $r_1 = \{$'US','orange','Yes','1100','120'$\}$ in $\mathcal{T}$ covers $s_2[1] = \{$'US', 'Yes'$\}$ in s_2, $s_4[1] = \{$'1250'$\}$ in s_4 and $s_5[1] = \{$'US', 'orange', '120'$\}$ in s_5. We set the flags of the 3 tuples as true and select r_1 into the unified synopsis. Iteratively, we select 7 records that can cover all tuples of all the query synopsis as shown in Figure 4(a).

Suppose there are N records in $\mathcal{T}$, and M tuples in all the synopses of τ queries, i.e., $M = \sum_{i=1}^{\tau} |s_i|$. Since the algorithm needs to enumerate each record in $\mathcal{T}$ and for each record checks each tuple in the query synopsis, the complexity of this algorithm is $\mathcal{O}(MN)$.

4.3.2 Greedy Algorithm

The scan-based method may select many unnecessary records in the unified synopsis. To address this problem, we propose a greedy method to reduce the synopsis size. For each record r_k in $\mathcal{T}$, it may cover multiple tuples in different queries. We call the number of tuples that r_k covers as the *coverage* of r_k. We compute the coverage of each record, and greedily select the records with the largest coverage until all the flags are set as true. Note that a record r_k exactly covers one tuple $s_i[j]$ for each synopsis s_i. In other words, there is a many-to-one mapping from a record to a synopsis tuple. Thus the coverage of a record is among $\tau, \tau-1, \cdots, 1$. So in the first step, we scan the records in $\mathcal{T}$, identify the records with the coverage of τ, and put them into the unified synopsis, and update the flags of tuples in query synopses. In next step, we scan the records in $\mathcal{T}$ again and identify the tuples with the coverage of $\tau - 1$, and put them into the unified synopsis. Iteratively, we generate the synopsis.

For example, consider the example in Table 1. We scan the table 3 times. In the first iteration, we compute the *coverage* of each record. The first record $r_1 = \{$'US','orange', 'Yes','1100','120'$\}$ in $\mathcal{T}$ covers $\{$'US', 'Yes'$\}$ of q_2, $\{$'1250'$\}$ of q_4 and $\{$'US', 'orange', '120'$\}$ of q_5, and the coverage is 3. The coverage of r_2 is 1 and that of r_3 is 2. We find r_1, r_4 and r_7 with *coverage* 3 in the first iteration. Thus we first select them. Iteratively, we select 6 records, which has the same size as the optimal synopsis as shown in Figure 4(b).

The algorithm iterates at most τ times. For each iteration, we will scan the table to find the records with the most uncovered tuples in different queries. For each record, we should test how many tuples it covers. So the complexity of the greedy algorithm is $\mathcal{O}(\tau MN)$.

4.4 Answer New Queries and Update the Synopsis

Considering a new query q, if its columns are contained by a QCS π_i, we use $\mathcal{S}$ to answer the query using the same rewritten query q_i' within the bound δ; otherwise, we cannot use the synopsis $\mathcal{S}$ to answer the query and instead we use the original data to answer the query.

If there are some QCSs that cannot be answered by the synopsis, we need to update the synopsis $\mathcal{S}$. However, it is expensive to update $\mathcal{S}$ online. To address this issue, we update the synopsis in a delay manner, for example, when the server is not busy, we update all QCSs that cannot be answered by the synopsis. Formally, for each such QCS π_j, we generate its synopsis $\mathcal{S}_j$, merge $\mathcal{S}_j$ with the synopsis $\mathcal{S}$, and update $\mathcal{S}$ using the merged results as discussed in Section 4.3.

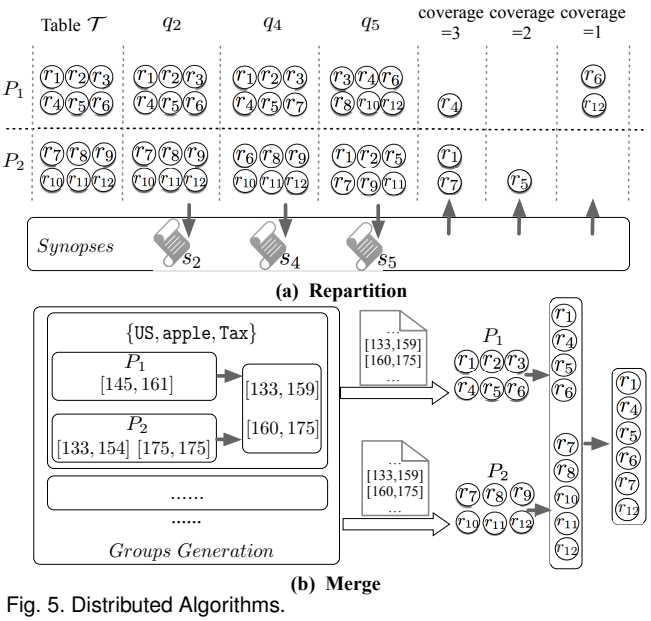


Fig. 5. Distributed Algorithms.

5 Distributed Algorithms

We study how to extend our method to support big data. We first study how to generate the same unified synopsis as the greedy algorithm in Section 5.1. As the method is rather expensive, we propose an efficient merge-based algorithm to generate the unified synopsis in Section 5.2.

5.1 Repartition-based Method

We first generate query synopsis s_i for each query q_i and then combine them to generate the unified synopsis.

Generating Query Synopsis s_i. (1) For the query with categorical columns only, we first select the distinct tuples on these categorical columns in each partition and then merge them together. We can prove that this method can correctly generate the query synopsis. (2) For the query with numerical columns only, we need to sort the numerical values. To sort the values, we need to repartition the records to put the close numerical values into the same partition, generate the numerical groups in each partition and then collect them together. (3) For the query with both numerical and categorical columns, we need to reparation the records based on the distinct tuples on the categorical columns, and put the records with the same distinct tuple into the same partition. Then we generate the synopsis in each partition and collect them together. For cases (2) and (3), we need to repartition the records, which is rather expensive.

Generating Unified Synopsis $\mathcal{S}$. We distribute the query synopsis s_i to each partition. Then, we select the records from each partition with the largest coverage τ and merge them together. We update the tuple flag in the query synopsis if the tuple is covered by the selected records and distribute the updated information to each partition. Next we select the record from each partition with the coverage $\tau - 1$ and merge them together. We repeat the above steps until all the tuples in each query synopsis are covered.

For example, in Figure 5(a), suppose that the table $\mathcal{T}$ is stored on two partitions P_1 and P_2. P_1 stores records r_1 to r_6 and P_2 stores records r_7 to r_{12}. We only consider queries q_2, q_4 and q_5 after eliminating redundant synopses. For q_2 with categorical columns only, we generate local synopsis on P_1 and P_2 as $\{\{$('US','Yes'), 3$\}$, $\{$('US','No'), 2$\}$, $\{$('CN','Yes'),

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2877362, IEEE Transactions on Knowledge and Data Engineering

1}, and {('FR','Yes'), 2}, {('US','Yes'), 1}, {('US','No'), 2}, {('CN','Yes'), 1}}. Then we merge them to generate synopsis s_2. Consider a more general case q_5 with both categorical and numerical columns. We repartition the records in $\mathcal{T}$ based on the categorical values in the Figure 3(b). We store 6 records with {'US', 'apple'} on P_1. We store the other 6 records on P_2. Then we sort the numerical values and generate 2 groups on P_1 and 3 groups on P_2. in Figure 5(a). We distribute s_2, s_4 and s_5 to each of the partitions. We select r_1 and update ('US','Yes') in s_2, (1250) in s_4 and ('US','orange',120) in s_5 as 'covered'. Then we find the next record with the maximum coverage. Iteratively, we select 6 records which has the same size as the optimal synopsis.

5.2 Merge-Based Method

The repartition-based method is rather expensive because (1) it requires to repartition the data, leading to huge numbers of data transmission, and (2) it needs to generate the query synopsis and distributes them to every partition. To address these issues, we propose a merge-based algorithm, which generates the local synopsis on each partition and then merges them to generate the global synopsis.

We first consider the queries with categorical columns only. We use the greedy algorithm to generate the local synopsis on each partition. Then we collect the local synopses and merge them by running the greedy algorithm to generate the global synopsis. However, for queries with numerical columns, it is hard to merge the local queries, because different partitions may generate different groups. Considering the example in Section 5.1, when we group the values in Tax for category {'US', 'apple'} for q_5, if one partition has one group [145,161] while another partition has two groups [133, 154] and [175, 175], they cannot be merged because [133, 154] and [145, 161] have overlap but combing them, i.e., [133, 161], is invalid as $\frac{161-133}{133} > 0.2$. So it is hard to merge the numerical groups.

To merge the groups, we need to use the same grouping strategy for different partitions. To achieve this goal, for a query with numerical columns, we first generate the local groups on each partition, and then refine them to generate the global groups and ask each partition to generate query synopsis based on the groups. Note each partition generates a set of groups, which must contain the minimum and maximum value. Thus we can get the global minimum value v_0 and maximum value v_N from these groups. Then we generate the global groups as follows. First, we initialize an empty global group set $\Phi = \varnothing$. We initially add a group as $[v_0, (1+\delta)v_0]$ and check that if there exists a local group g_l such that $g_l \cap [v_0, (1+\delta)v_0] \neq \varnothing$. If yes, we add $[v_0, (1+\delta)v_0]$ to Φ; otherwise we discard $[v_0, (1+\delta)v_0]$. Then we add a group $[(1+\delta)v_0 + 1, ((1+\delta)v_0+1)(1+\delta) + 1]$ and check if it overlaps with local groups. Iteratively we generate all global groups. We prove that the number of groups generated is at most twice of the optimal group number.

Lemma 8. The number of groups found by the merge-based method is at most twice of the optimal group number.

Based on the groups, we can generate the local synopsis for each query. Then we collect the local synopses and merge them by the greedy algorithm. As the local synopsis uses the same group, we can easily merge them.

Considering the above example, in Figure 5(b), we group the values in Tax on category {'US', 'apple'} for q_5. P_1 has one group [133,145] while P_2 has two groups [140, 161] and [175, 175]. We can generate groups between minimum 133 and maximum 175. First, we generate [133,159] and find $[133, 159] \cap [133, 145] = [133, 145] \neq \varnothing$. Then we find $[160, 175] \cap [140, 161] \neq \varnothing$. So we finally get two groups [133,159] and [160,175]. Next we generate local synopsis on P_1 , i.e., r_1, r_4, r_5, r_6 and local synopsis on P_2., i.e., r_7, r_8, r_9,r_{10}, r_{11}, r_{12}. We use the greedy method on them to generate a unified synopsis with $r_1, r_4, r_5, r_6, r_7, r_{12}$ which has the same size as the optimal synopsis.

6 Experimental Evaluation

We compared our method BAQ with state-of-the-art SAQ (we implemented BlinkDB with stratified sampling [1] and Seek [12] using sample and index), DAQ [33], and Histogram [31]. As BlinkDB had verified that SAQ was better than Sketch, Wavelet, we did not evaluate them.

6.1 Setting

Datasets. We used two datasets. The first is Microsoft real-life production workload dataset MS with 30 columns (20 categorical columns and 10 numerical columns) [2]. The dataset contained the statistics of Bing search and was used for query analysis. 77.5% of queries accessed 7 to 20 columns, and there were less than 5% queries change within a month. The second is the well-known TPC-H dataset and we use the table of orders with 4 columns (3 categorical and 1 numerical). The statistics were shown in Table 4.

Queries. In MS dataset, we generated 1000 queries with 100 QCSs, including 200 queries with only 1 categorical column, 200 queries with 2 categorical columns, 100 queries with 3 categorical columns, 10 queries with 1 numerical columns, 10 queries with 2 numerical columns, 200 queries with 1 category + 1 numerical columns, 200 queries with 2 category + 1 numerical columns, and 80 queries with 3 categorical + 1 numerical columns. In TPC-H dataset, we generated 250 queries with 6 QCSs, including 100 queries with 1 categorical column, 15 queries with 2 categorical columns, 120 queries with 1 categorical + 1 numerical columns, and 15 queries with 2 categorical+ 1 numerical columns.

Metrics. We compared three methods (DAQ, BAQ, SAQ) from four aspects. (1) Answer Error. (2) Synopsis Size (#Samples). (3) Online Query Processing Time (in milliseconds). (4) Offline Synopsis Generation Time (in seconds). We ran each method 20 times and reported the average results.

Setting. We also evaluated our method on Spark. We used a cluster with 20 nodes. Each node had a memory of 112GB and the cluster had totally 2TB memory and 152 cores. All the dataset was resident in main memory.

6.2 BAQ vs SAQ

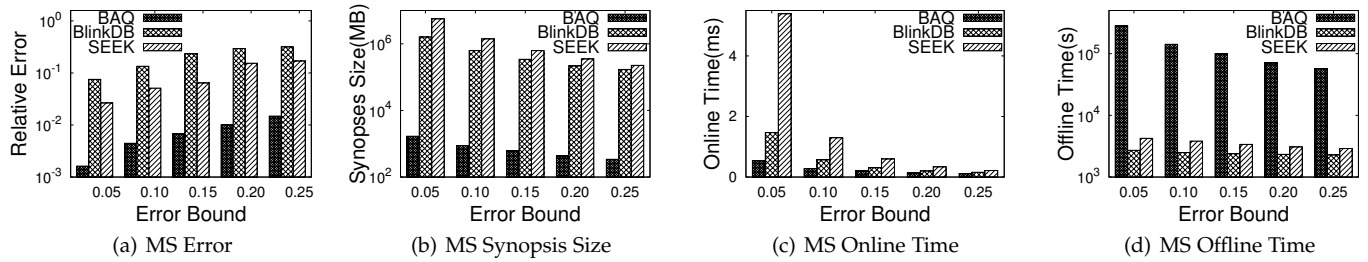
6.2.1 Varying Error Bound

We compared SAQ and BAQ by varying the error bounds. For SAQ, we implemented two variants BlinkDB and Seek. For BlinkDB, we set its confidence as 99.9%. For Seek, we set the sample size as $\frac{\sqrt{N}}{\delta^2}$. We used SAQ to denote BlinkDB/Seek. For BAQ, we used the greedy algorithm to generate synopsis. Figure 6 showed the results.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2877362, IEEE Transactions on Knowledge and Data Engineering

Dataset	row	column	size	average distinct value	min/max numerical	Description
MS-Dataset	23M	30	125.75G	417	0/14399999	Microsoft Product
TPC-H order.tbl	150M	4	15G	20	811.73/568754.48	Standard TPC

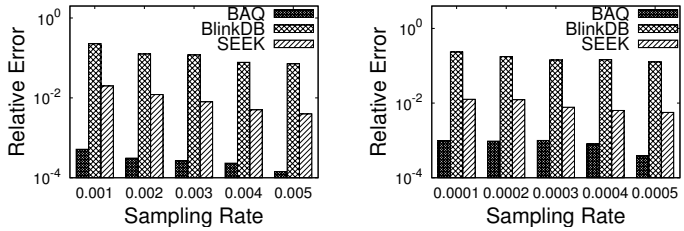
TABLE 4
Two Datasets.



(a) MS Error (b) MS Synopsis Size (c) MS Online Time (d) MS Offline Time

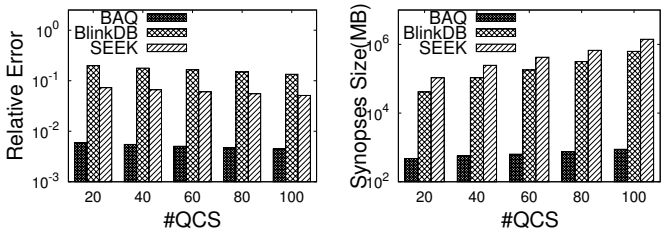
(e) TPC Error (f) TPC Synopsis Size (g) TPC Online Time (h) TPC Offline Time

Fig. 6. BAQ vs SAQ: Varying Error Bound.



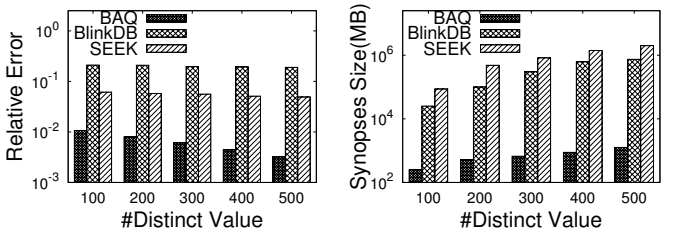
(a) MS Error (b) TPC Error

Fig. 7. BAQ vs SAQ: Varying Sampling Rate.



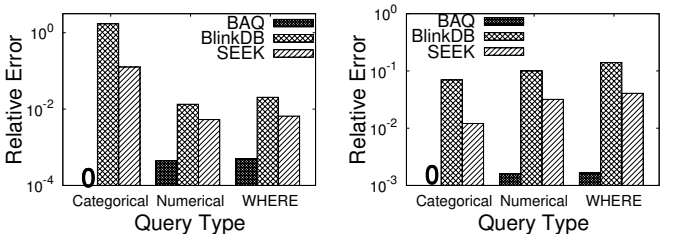
(a) Error (b) Synopsis Size

Fig. 8. BAQ vs SAQ: Varying QCS on MS.



(a) Error (b) Synopsis Size

Fig. 9. BAQ vs SAQ: Varying Distinct Values on MS.



(a) MS Error (b) TPC Error

Fig. 10. BAQ vs SAQ: Different Type of Queries.

(1) Error Analysis. BAQ had much smaller relative error, even 10-100× better than SAQ. For example, with different error bounds, BAQ had 0.1%-1% error while SAQ had 10% error. The reasons were two-fold. First, BAQ had a confidence of 100% for most cases. Second, BlinkDB worked well for normal distribution but cannot for other distributions. BlinkDB and BAQ had smaller error on TPC-H than MS, because TPC-H followed the uniform distribution and it was easy to meet the error bounds but MS did not follow the uniform distribution. With the increase of error bounds, the error also increased but the error was always smaller than the given bound, because the bound was estimated for worst cases, and in real cases, the error was much smaller. Seek had smaller errors than BlinkDB as it used indexes to support the queries with fewer answers.

(2) Synopsis Size. BAQ had much smaller synopsis size than SAQ, even 100-1000× smaller than SAQ. For example, with different error bounds, the synopsis sizes of BAQ were less than 1GB while those of SAQ were more than 100GB. The main reasons were two-fold. First, SAQ generated a synopsis for each QCS while BAQ generated a unified synopsis. Second, BAQ judiciously selected the synopsis to cover the queries while SAQ used sampling-based methods and cannot select high-quality samples. In addition, with the increase of the error bounds, the synopsis size decreased, as they could select fewer samples to meet the error bound. The downward trend on TPC-H was faster than that on MS, as TPC-H had fewer columns and they selected fewer samples. Seek had larger sizes than BlinkDB as it selected more samples to generate the synopsis.

(3) Online Query Time. BAQ was faster than SAQ, as BAQ had smaller synopsis size than SAQ. BAQ was rather efficient and could answer a query within 1ms. With the increase of the error bounds, the online query time decreased, because they used smaller synopses to answer a query. They have better performance on TPC-H because TPC-H only had 4 columns. Seek had longer latency than BlinkDB, as Seek separately answered different queries and used more samples.

(4) Offline Synopsis Generation Time. BAQ took more time than SAQ to generate the synopsis because BAQ needed

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2877362, IEEE Transactions on Knowledge and Data Engineering

to select high-quality samples while SAQ used sampling methods and might not select high-quality samples. As the synopsis generation was offline, it was acceptable.

Summary. BAQ outperformed SAQ in terms of both error bounds, synopsis sizes and online query time.

Due to the space constraints, we focused on error bounds and synopsis size in the following experiments.

6.2.2 Varying Sampling Rate

We compared SAQ and BAQ by varying the same sampling rate from 0.001 to 0.005. For each sampling rate, we found an appropriate error bound for BAQ by binary searching the error bounds between 0.01 and 0.5 to meet the sampling rate. We used the greedy algorithm to generate the synopsis. Figure 7 showed the results. As the sampling rates were the same, they had similar synopsis size. So we only showed the results on relative error. Firstly, BAQ still significantly outperformed SAQ in each sampling rate. Secondly, with increase of sampling rates, the relative errors decreased because more synopsis could be used to answer queries. BAQ had much better result quality because it had 100% confidence on the error bounds.

6.2.3 Varying #QCS

We set the error bound as 0.1 and compared SAQ and BAQ by varying the number of QCSs from 20 to 100. We used the greedy algorithm to generate synopses offline. Figure 8 showed the result. We had the following observations. Firstly, when the number of QCS increased, the error of BAQ and SAQ increased a little. Thus BAQ and SAQ were not sensitive on the number of QCSs in terms of errors, because both of them generated synopsis for all the QCS within the error bound. Secondly, the synopsis size of BAQ increased slower than that of SAQ with the increase of QCS. Thus SAQ was more sensitive on the number of QCSs in terms of synopsis size; however the number of QCSs had no significant effect on BAQ, as BAQ generated a unified synopsis rather than generated synopses for all QCSs in SAQ.

6.2.4 Varying #Distinct Values in Columns

We set the error bound as 0.1 and compared SAQ and BAQ by varying the number of average distinct values from 100 to 500. We used the greedy algorithm to generate a synopsis offline. Figure 9 showed the result. Firstly, with the increase of distinct values, both BAQ and SAQ generated larger synopsis because they depended on the number of distinct values. However, the synopsis size of BAQ and SAQ linearly increased when the number of distinct value increased. This was because when the number of distinct value increased, BAQ should select more tuples in the unified synopsis while SAQ should consider more groups. Thus BAQ and SAQ worked very well for large numbers of distinct values. Secondly, with the increased number of distinct values, the errors decreased because they selected larger synopsis to answer queries. BAQ was more sensitive on error than SAQ when the number of distinct values varied, because BAQ generated smaller synopsis while SAQ selected more samples to support the confidence within the error bound.

6.2.5 Varying Query Type

We compared SAQ and BAQ on different types of queries: categorical only, queries without numerical columns in the where clause, and queries with numerical columns in where clause. We used the greedy algorithm to generate a synopsis offline. Figure 10 showed the results. (1) BAQ could exactly answer categorical only queries with no error but SAQ always had a large error, because BAQ used scale factor to keep the frequencies of all categories but SAQ failed to get the exact number. (2) BAQ got smaller error than SAQ on queries with numerical columns, because BAQ could achieve a higher quality using the scale factor with 100% confidence. Thus, BAQ had much higher quality than SAQ on all queries. (3) Both of BAQ and SAQ had a higher error when there existed numerical columns in the where clause, because BAQ and SAQ could not accurately estimate the number of records that satisfied the selection conditions of the queries. Fortunately, the result indicated that the additional error caused by partially satisfied groups was small.

6.3 BAQ vs DAQ

Both BAQ and DAQ gave an approximate result within a given error bound with a 100% confidence. We compared BAQ and DAQ by varying error bounds. As DAQ used the absolute error, we varied the error bounds from 2^{10} to 2^{30}. As DAQ cannot support WHERE and GROUPBY, we compared BAQ and DAQ for queries with numerical columns only.

6.3.1 Real Dataset

Figure 11 showed the result on the two real datasets. We had the following observations. (1) BAQ ran 100× faster than DAQ for larger error bounds. This was because BAQ used a small-size synopsis while DAQ should compute the high-order bits of all the records. Thus DAQ had limited performance improvement compared with the exact algorithms. (2) BAQ had smaller errors than DAQ. This was because DAQ ignored the impact of the low-order bits so that it would get a higher error than BAQ, especially for larger error bounds. (3) DAQ had smaller errors on MAX than on AVG because it captured the maximal values using the high-order bits.

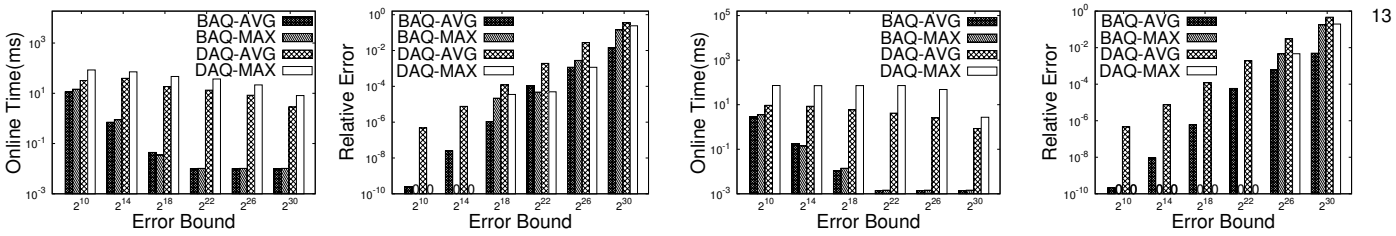
6.3.2 Zipf Distribution

As DAQ was better than SAQ and here we only compared with DAQ. To compare the error bounds of BAQ and DAQ on data with heavily skewed distribution, we generated datasets with Zipf distribution by setting the parameter of Zipf as 1.5. Figure 12 showed the results. We had following observations. (1) BAQ and DAQ had smaller errors for MAX, since both of them could support maximum values well. BAQ used the samples in a tight range to support maximum value, while DAQ used the high-order bits to find the maximum value. Thus both of the two methods could support heavily skewed data. (2) BAQ had smaller errors than DAQ and the reason was similar to the case on real datasets. (3) BAQ was faster than DAQ as DAQ needed to use all records.

6.4 BAQ vs Histogram

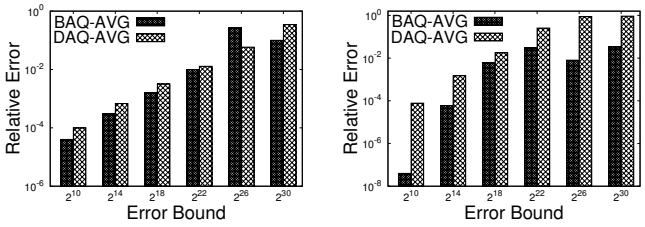
Both BAQ and Histogram could divide the numerical values into buckets within bounded error. We set the error bound as 0.1 and compared BAQ and state-of-the-art Histogram by varying the number of QCSs from 20 to 100. BAQ and Histogram had the same relative error but BAQ used less storage than Histogram, because BAQ constructed a unified synopsis for all the QCSs while Histogram should construct

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2877362, IEEE Transactions on Knowledge and Data Engineering



(a) MS Online Time (b) MS Relative Error (c) TPC-H Online Time (d) TPC-H Relative Error

Fig. 11. BAQ vs DAQ: Varying Error Bounds.



(a) Zipf 1.5 MS (b) Zipf 1.5 TPC-H

Fig. 12. BAQ vs DAQ: Zipf Distributions.

one synopsis for each of the QCSs. Table 5 showed the result. With the increase of QCSs, the size of BAQ synopsis increased slowly while the size of Histogram synopsis increased quickly. Further, Histogram could not support categorical query well comparing with BAQ.

#QCS	20	40	60	80	100
BAQ	491.3M	554.8M	658.2M	718.7M	790.5M
Histogram	1.3G	3.4G	5.9G	10.6G	14.9G

TABLE 5
Synopsis Size of BAQ and Histogram.

#Columns	6	12	18	24	30
Synopsis Size(MB)	392.6	548.2	650.0	730.5	790.5
Construction Time(s)	196	235	262	285	300

TABLE 6
Varying Column Number.

6.5 Evaluating Synopsis Generation in BAQ

6.5.1 Synopsis Generation: Scan vs Greedy

We evaluated the synopsis generation algorithms, i.e., scan-based algorithm and greedy algorithm. We varied the error bound from 0.05 to 0.25 to evaluate the synopsis size, the offline synopsis generation time, and the error bounds. Figure 13 showed the results. We had the following observations. Firstly, the scan-based algorithm always took less time than the greedy algorithm, because the scan-based algorithm scanned the table once to generate the unified synopsis while the greedy algorithm needed to scan the table multiple times. Secondly, the greedy algorithm generated much smaller synopsis than the scan-based algorithm because the greedy algorithm aimed to select synopsis with the minimum number of records to cover the query synopses. Thirdly, the error bounds of the scan-based algorithm and the greedy algorithm were similar, because the scan-based algorithm selected more tuples while the greedy algorithm covered more query synopses.

6.5.2 Distributed Algorithms: Repartition vs Merge

We evaluated the repartition-based and merge-based methods on Spark to evaluate the offline synopsis generation time and synopsis size. The result was shown in Figure 14. We had following observations. Firstly, our two algorithms were very fast on Spark. Secondly, the merge-based algorithm performed faster than the repartition-based algorithm on Spark, because the repartition-based algorithm took much time to shuffle the data but the merge-based algorithm just needed to generate synopsis on each node and combined them together. Thirdly, the size of synopsis generated by the merge-based algorithm was a little bigger than the repartition-based algorithm, because the merge-based algorithm generated more groups on numerical columns. Fourthly, the error bounds of the merge-based algorithm was a bit larger than the repartition-based algorithm, as the merge-based algorithm only found local optimal results but did not find the global optimal results.

6.5.3 Varying Number of Columns

We evaluated the performance on the MS dataset by varying the number of columns from 6 columns (4 categorical and 2 numerical) to 30 columns (20 categorical and 10 numerical). We reported the results of the merge-based algorithm in Section 5.2 and set the error bound as 0.1. Table 6 showed the construction time and synopsis size. We had the following observations. (1) The construction time increased slowly as the column number increased, because the time cost depended on the number of tuples but was not sensitive to column numbers. (2) The synopsis size increased with the increasing of column numbers, because it required to cover more tuples of more columns. But the increasing rate was small and our method still worked well for many columns, e.g., 30. Thus even though there were multiple columns, our method could still generate high-quality synopsis.

7 Conclusion

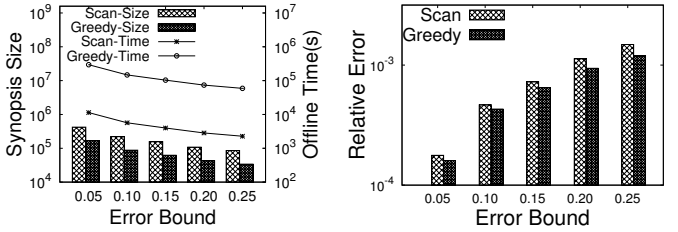
We proposed a bounded approximate query processing framework, which supported most SQL aggregation queries with lower error bounds. BAQ first selected high-quality synopsis for each query and then constructed a unified synopsis by covering each query synopsis. We proved that selecting the optimal unified synopsis was NP-hard and devised effective algorithms. We developed distributed algorithms to generate the unified synopsis in distributed environments. Experimental results showed that BAQ had lower errors and smaller synopsis size compared with state-of-the-arts.

Acknowledgement. This paper was supported by 973 Program of China (2015CB358700), NSF of China (61632016, 61521002, 61472198, 61661166012), BHJ14L010, Huawei, and TAL education. Yong Zhang and Guoliang Li are corresponding authors.

References

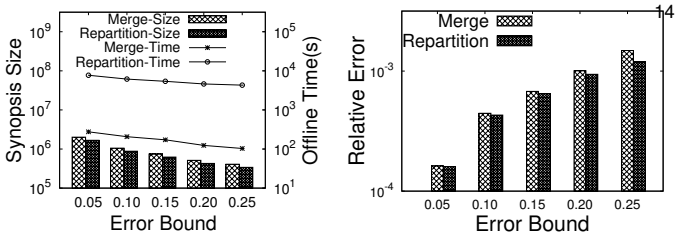
[1] Blinkdb homepage. http://blinkdb.org/.
[2] Taming big wide tables. http://acmsocc.github.io/2015/posters/socc15posters-final24.pdf.
[3] S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. I. Jordan, S. Madden, B. Mozafari, and I. Stoica. Knowing when you're wrong: building fast and reliable approximate query processing systems. In *SIGMOD*, pages 481–492, 2014.
[4] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Eurosys*, pages 29–42, 2013.
[5] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *SIGMOD*, pages 539–550, 2003.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2018.2877362, IEEE Transactions on Knowledge and Data Engineering



(a) Time and Size (b) Error

Fig. 13. Scan vs Greedy.



(a) Time and Size (b) Error

Fig. 14. Repartition vs Merge.

[6] D. P. Bertsekas and J. N. Tsitsiklis. Introduction to probability : Problem solutions. 2008.

[7] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *VLDB J.*, 10(2-3):199–223, 2001.

[8] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. In *NSDI*, pages 313–328, 2010.

[9] G. Cormode, A. Deligiannakis, M. N. Garofalakis, and A. McGregor. Probabilistic histograms for probabilistic data. *PVLDB*, 2(1):526–537, 2009.

[10] 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.

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

[12] B. Ding, S. Huang, S. Chaudhuri, K. Chakrabarti, and C. Wang. Sample + seek: Approximating aggregates with distribution precision guarantee. In *SIGMOD*, pages 679–694, 2016.

[13] M. N. Garofalakis and J. Gehrke. Querying and mining data streams: You only get one look. In *VLDB*, 2002.

[14] M. N. Garofalakis and P. B. Gibbons. Wavelet synopses with error guarantees. In *SIGMOD*, pages 476–487, 2002.

[15] S. Guha and B. Harb. Wavelet synopsis for data streams: minimizing non-euclidean error. In *SIGKDD*, pages 88–97, 2005.

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

[17] T. Hesterberg. What teachers should know about the bootstrap: Resampling in the undergraduate statistics curriculum. In *The American Statistician*, volume 69, pages 371–386, 2015.

[18] Y. E. Ioannidis. Approximations in database systems. In *ICDT*, page 1630, 2003.

[19] Y. E. Ioannidis. The history of histograms (abridged). In *VLDB*, pages 19–30, 2003.

[20] S. Kandula, A. Shanbhag, A. Vitorovic, M. Olma, R. Grandl, S. Chaudhuri, and B. Ding. Quickr: Lazily approximating complex adhoc queries in bigdata clusters. In *SIGMOD*, pages 631–646, 2016.

[21] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

[22] L. V. S. Lakshmanan, N. Leone, R. B. Ross, and V. S. Subrahmanian. Probview: A flexible probabilistic database system. *ACM Trans. Database Syst.*, 22(3):419–469, 1997.

[23] L. V. S. Lakshmanan and F. Sadri. Uncertain deductive databases: A hybrid approach. *Inf. Syst.*, 22(8):483–508, 1997.

[24] F. Li, B. Wu, K. Yi, and Z. Zhao. Wander join: Online aggregation for joins. In *SIGMOD*, pages 2121–2124, 2016.

[25] Y. Matias, J. S. Vitter, and M. Wang. Dynamic maintenance of wavelet-based histograms. In *VLDB*, pages 101–110, 2000.

[26] M. Muralikrishna and D. J. DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In *SIGMOD*, pages 28–36, 1988.

[27] F. Olken and D. Rotem. Simple random sampling from relational databases. In *VLDB*, pages 160–169, 1986.

[28] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie. Online aggregation for large mapreduce jobs. *PVLDB*, 4(11):1135–1145, 2011.

[29] A. Pol and C. Jermaine. Relational confidence bounds are easy with the bootstrap. In *SIGMOD*, pages 587–598, 2005.

[30] V. Poosala and V. Ganti. Fast approximate answers to aggregate queries on a data cube. In *SSDBM*, pages 24–33, 1999.

[31] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. In *SIGMOD*, pages 294–305, 1996.

[32] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. In *SIGMOD*, pages 294–305, 1996.

[33] N. Potti and J. M. Patel. DAQ: A new paradigm for approximate query processing. *PVLDB*, 8(9):898–909, 2015.

[34] C. Qin and F. Rusu. PF-OLA: a high-performance framework for parallel online aggregation. *Distributed and Parallel Databases*, 32(3):337–375, 2014.

[35] V. P. S. Acharya, P. B. Gibbons and S. Ramaswamy. The aqua approximate query answering system. In *SIGMOD*, page 28(2), 1999.

[36] G. D. S. Chaudhuri and V. Narasayya. Optimized stratified sampling for approximate query processing. In *TODS*, 2007.

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

[38] D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[39] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *SIGMOD*, pages 193–204, 1999.

[40] H. Wang and K. C. Sevcik. Utilizing histogram information. In *Centre for Advanced Studies on Collaborative Research*, page 16, 2001.

[41] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *SIGMOD*, pages 469–480, 2014.

[42] Y. Yan, L. J. Chen, and Z. Zhang. Error-bounded sampling for analytics on big sparse data. *PVLDB*, 7(13):1508–1519, 2014.

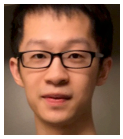
[43] K. Yi, F. Li, M. Hadjieleftheriou, G. Kollios, and D. Srivastava. Randomized synopses for query assurance on data streams. In *ICDE*, pages 416–425, 2008.

[44] K. Zeng, S. Gao, B. Mozafari, and C. Zaniolo. The analytical bootstrap: a new method for fast error estimation in approximate query processing. In *SIGMOD*, pages 277–288, 2014.

Kaiyu Li received the bachelor's degree from the Department of Computer Science and Technology, Harbin Institute of Technology, China. He is currently working toward the Master degree in the Department of Computer Science, Tsinghua University, Beijing, China. His research interests include approximate query processing, data integration and crowdsourcing.

Yong Zhang is an associate professor of Research Institute of Information Technology at Tsinghua University. He received his BSc degree in Computer Science and Technology in 1997, and PhD degree in Computer Software and Theory in 2002 from Tsinghua University. From 2002 to 2005, he did his Postdoc at Cambridge University, UK. His research interests are data management and data analysis.

Guoliang Li is currently working as an associate professor in the Department of Computer Science, Tsinghua University, Beijing, China. He received his PhD degree in Computer Science from Tsinghua University, Beijing, China in 2009. His research interests mainly include data cleaning and integration, spatial databases and crowdsourcing.



Wenbo Tao received the bachelor's degree from the Department of Computer Science, Tsinghua University, Beijing, China. He is currently a PHD student in MIT, Boston, U.S. His research interests include approximate query processing, data visualization, data integration.

Ying Yan is currently working as a Lead Researcher in Cloud Computing Group, MSRA. She got her PhD degree in computer science from Fudan University. Her current research interest includes big data analytic, database optimization and Blockchain technology.