Demonstration of ScroogeDB: Getting More Bang For the Buck with Deterministic Approximation in the Cloud

Saehan Jo
Cornell University, NY, USA
sj683@cornell.edu

Jialing Pei
Cornell University, NY, USA
jp2475@cornell.edu

Immanuel Trummer
Cornell University, NY, USA
itrummer@cornell.edu

ABSTRACT

We demonstrate ScroogeDB which aims at minimizing monetary cost of processing aggregation queries in the Cloud. It runs on top of a Cloud database that offers pay-as-you-go query processing where users pay according to the number of bytes processed. ScroogeDB exploits deterministic approximate query processing (DAQ) to achieve monetary savings. That is, ScroogeDB provides deterministic bounds, i.e., bounds that contain the true value with a 100% probability. ScroogeDB creates small synopses of the database and uses these synopses to answer aggregation queries. By rewriting a query on base tables into a query on smaller synopses, we significantly reduce the amount of processed data. We do not pre-compute synopses in advance of an analysis session. Instead, we generate them on the fly, interleaving synopsis generation with query execution.

In our demonstration, we show that our system realizes impressive monetary savings with little precision loss. We run our system on top of the Google BigQuery Cloud platform and provide users with a graphical interface that visualizes deterministic bounds. The graphical interface also provides information regarding the generated synopses and how they contribute to monetary savings.

PVLDB Reference Format:
Saehan Jo, Jialing Pei, Immanuel Trummer. Demonstration of ScroogeDB: Getting More Bang For the Buck with Deterministic Approximation in the Cloud. *PVLDB*, 13(12): 2961-2964, 2020.
DOI: https://doi.org/10.14778/3415478.3415519

1. INTRODUCTION

With the proliferation of Cloud-based query processing services like Google BigQuery or Amazon Redshift, very large data sets have become more readily accessible for data analysis. A novel pricing model available for Cloud-service users is the pay-as-you-go policy where users pay solely based on the amount of data processed and stored in the Cloud. In this scenario, we want to minimize the amount of data read per query to minimize monetary fees for end-users. ScroogeDB aims at minimizing the number of bytes processed via deterministic approximate query processing.

Prior work on approximate query processing (AQP) typically uses sampling to generate confidence bounds on query aggregates [1]. Confidence bounds can be hard to interpret for lay users [6]. Sampling is problematic if aggregates are sensitive to outliers (e.g., maxima and minima). Those drawbacks have recently motivated DAQ [6], aimed at producing hard bounds on query aggregates. Prior work on DAQ focuses on simple (single table) queries [4, 6] or time series data [2]. Another recent work, BitGourmet [3], introduces a specialized execution engine that operates on bit-sliced data to produce deterministic bounds. All prior work focuses on minimizing execution time, rather than monetary execution fees. On the other hand, ScroogeDB is non-intrusive (i.e., runs on top of existing Cloud databases) and focuses on minimizing monetary fees.

ScroogeDB operates as a layer on top of a Cloud processing platform offering pay-as-you-go pricing. It is non-intrusive since it does not require access to the internals of the Cloud database. When a user issues a query, ScroogeDB rewrites the query into a semantically equivalent query on data synopses and executes it instead. Each data synopsis is a table that summarizes a portion of the database (covering one or multiple base tables). These synopses are much smaller in size than base tables, and thus, using synopses rather than base tables reduces monetary fees. ScroogeDB chooses the synopsis to use to answer a specific query based on a precision model. The precision model estimates the result precision when answering the current query using a specific synopsis. Given a user-defined precision constraint, ScroogeDB finds the synopsis with minimum processing cost among synopses that satisfy the precision constraint.

Prior AQP systems usually generate auxiliary data structures offline, prior to query executions. Their goal is to reduce run time, whereas our goal is to reduce monetary cost. Since we pay per byte processed, creating redundant synopses directly translates into unnecessary monetary fees. We have to ensure that each generated synopsis pays off. For this reason, ScroogeDB generates synopses in an online fashion. That is, we interleave synopsis generation (which is done in the background) with query execution. ScroogeDB does not require an example workload nor does it create pre-computed synopses. Instead, we only rely on (a handful of) queries from the ongoing analysis session to determine the query distribution in a probabilistic manner.

Based on that, ScroogeDB searches for a synopsis that benefits us not only for the current query but also in future

Proceedings of the VLDB Endowment, Vol. 13, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3415478.3415519

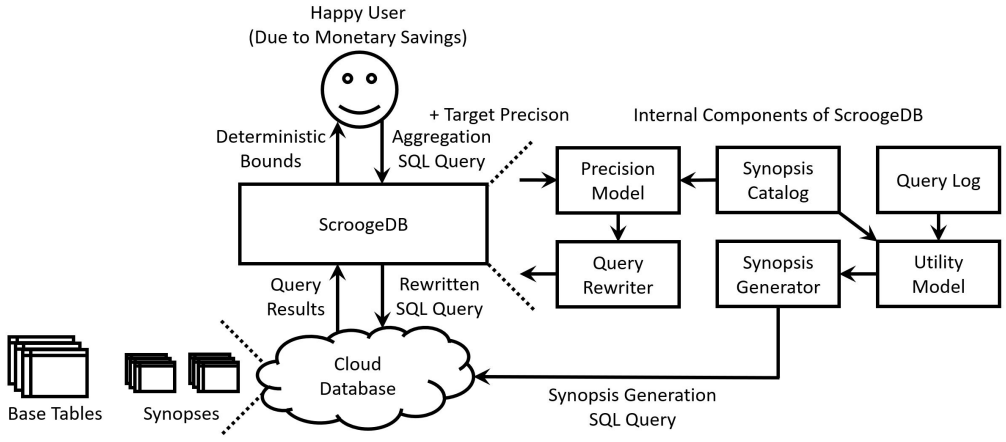

Figure 1: Overview of the ScroogeDB system.

query executions. ScroogeDB relies on a cost-based synopsis utility model to determine the optimal synopsis to generate. We formulate the problem of choosing the synopsis with maximum utility as an integer linear program.

In our demonstration, we run ScroogeDB on top of Google BigQuery and demonstrate its effectiveness in reducing monetary cost. ScroogeDB demonstrates impressive monetary savings with little precision loss on query results. Section 2 gives an overview of ScroogeDB, introducing its components. In Section 3, we provide details on our demonstration plan which includes an interactive user experience of our system.

2. PROBLEM MODEL

We introduce our problem model and relevant definitions. ScroogeDB supports aggregation queries with key-foreign key joins on a star schema. We currently support Count, Sum, Avg, Min, and Max aggregation functions and group by clauses as well as equality and inequality predicates.

2.1 Data Synopsis

A *Data Synopsis* is a table summarizing a portion of the underlying database. It can cover a subset of columns in the fact table as well as columns in dimension tables. To specify a synopsis, we select a subset of columns from the database. Then, the synopsis consists of all distinct value combinations appearing in the selected columns and their frequencies. In other words, a data synopsis resembles a materialized view created by the following SQL statement:

```
select C, Count(*) as cnt
from F
[join D on {foreign key constraint}]*
group by C;
```

where C denotes a subset of columns in the database, F is the fact table, and D denotes a dimension table. Note that there can be multiple dimension tables, each joined along the foreign key constraint.

Now, we introduce approximation to synopses. We can divide the value domain of a column so that distinct values are clustered into a smaller number of coarser value groups. Each value group can represent multiple consecutive values in the value domain. We call a column in a synopsis an *Approximate Column* if its value domain is divided into coarser value ranges. Otherwise, if the value domain of a column in a synopsis is at its finest granularity, we call it an *Exact Column*. This is when there is one value per value group. We use a tilde over a column symbol to denote an approximate column (i.e., $\tilde{c}$ indicates that a column c is approximate). The size of a synopsis depends on how many distinct value combinations appear in the selected columns. With approximate columns, we can reduce the number of distinct value combinations, and thus, shrink the size of a synopsis even further compared to when it only contains exact columns. For an approximate column, a synopsis stores the minimum and maximum values of each value group.

a	c	cnt
A	1	54
B	1	78
A	2	17
B	2	19
B	3	11
A	4	3
B	5	8

Exact Column c

a	$c_{\min}$	$c_{\max}$	cnt
A	1	2	71
B	1	2	97
B	3	5	19
A	4	4	3

Approximate Column $\tilde{c}$

Figure 2: Example of exact and approximate columns in a synopsis.

Example 1. Figure 2 depicts a case where the number of distinct value combinations reduces from seven to four. We divide the value domain $[1, 5]$ of an integer column c into two value ranges $[1, 3)$ and $[3, 5]$. The synopsis with the approximate column $\tilde{c}$ stores its minimum and maximum for each value range.

2.2 Problem Statement

We measure the *Precision* of deterministic bounds as the lower bound divided by the upper bound. In other words, we get l/u where l is the lower bound and u is the upper bound (assuming positive numbers for aggregates). We get a precision of one when the query result is exact (i.e., $l = u$) and a precision of zero when we have a trivial lower bound of zero.

The goal of *Monetary Fee Minimization via DAQ* is to minimize the monetary cost of pay-as-you-go query process-

ing in the Cloud. In our problem model, a user conducts an analysis session and executes queries on a database one after another. In this scenario, there is no example workload and the specifics of a query are not given until its execution. We exploit DAQ in our solution to achieve this goal. Given a target precision, we produce deterministic bounds with the desired level of precision as query results.

3. SYSTEM OVERVIEW

Figure 1 shows how ScroogeDB interacts with the underlying Cloud database to reduce monetary cost. When a user issues an aggregation query, ScroogeDB rewrites it into a semantically equivalent query on data synopses (*Query Rewriter* in Figure 1). Synopses are much smaller in size, compared to base tables, and thus, the rewritten query accesses less data than the original. As a result, ScroogeDB reduces monetary cost, as pay-as-you-go query processing charges users per byte processed.

We illustrate the internal process of ScroogeDB. First, given an aggregation query to execute, ScroogeDB identifies synopses (among all synopses stored in the Cloud) that can be used to answer the current query while satisfying the user-defined precision constraint (*Precision Model* in Figure 1). If there are multiple such synopses, we choose a synopsis that results in the minimal number of bytes processed. Otherwise, we consider generating a new data synopsis. ScroogeDB takes into account whether a synopsis can answer the current query as well as the expected utility for future queries in the ongoing analysis session (*Utility Model* in Figure 1). We consider two factors to estimate synopsis utility: 1) how likely the synopsis is to be used and 2) how much savings we get per usage. To predict upcoming queries, we use a probabilistic approach and model the query distribution of the current session. If the size of the smallest synopsis that can answer the current query is already comparable to that of base tables, there is no benefit in using a synopsis. In that case, we do not create a synopsis and execute the query without rewriting it.

3.1 Query Rewriting

ScroogeDB utilizes query rewrite rules to transform an aggregation query on base tables into a semantically equivalent form that operates on a synopsis. The high-level idea for query rewriting is to scale aggregated values in the synopsis based on the frequency counts. The following example illustrates rewriting a query using an exact column.

Example 2. Consider the synopsis with the exact column c in Figure 2 and the aggregation query `Sum(`c`) where` $a =$`'B'`. To compute a sum over c, we simply multiply each value in c with the frequency count and add them up to get the result. In other words, we get $1 \cdot 78 + 2 \cdot 19 + 3 \cdot 11 + 5 \cdot 8 = 189$. In summary, we rewrite the query as `Sum(`$c \cdot cnt$`) where` $a =$`'B'`.

Now, suppose we use the synopsis with the approximate column $\tilde{c}$ to calculate the summation over c.

Example 3. Consider the same query `Sum(`c`) where` $a =$`'B'`. ScroogeDB computes the deterministic lower bound by multiplying the frequency count to the minimum value for each row and adding them together. We get a lower bound of $1 \cdot 97 + 3 \cdot 19 = 154$. Similarly, we compute the deterministic upper bound using the maximum values as $2 \cdot 97 + 5 \cdot 19 = 289$. In short, we rewrite the query as `Sum(`$c_{\min} \cdot cnt$`) as lower,` `Sum(`$c_{\max} \cdot cnt$`) as upper where` $a =$`'B'`. Note that $c_{\min}$ and $c_{\max}$ are column names rather than aggregates.

3.2 Precision Model

If a synopsis only contains exact columns, we get the best possible precision of one (i.e., we compute exact results). When a query evaluation involves approximate columns, we still can get a lower bound on the precision. We can show that if the relevant approximate columns satisfy certain properties, it is guaranteed that the query result meets the desired level of precision. We describe this concept using the following example.

Example 4. Suppose we are executing the following query `Sum(`c`) where` $a =$`'B'` using the synopsis with the approximate column $\tilde{c}$ in Figure 2. Each value range of $\tilde{c}$ has the property that the ratio of the lower bound to the upper bound is greater than or equal to 0.5. That is, we get $1/2 = 0.5$ for the first value range and $3/5 = 0.6$ for the second value range. Then, it is guaranteed that the deterministic bounds of the summation give us a precision better than or equal to 0.5. For this example, we get a precision of $154/289 \approx 0.53$ from the deterministic bounds we computed in Example 3.

This is true for the case when an approximate column is used as an aggregated column or a constrained column in an inequality predicate. For other cases (e.g., equality predicate or group by clauses), we resort to exact columns in the synopsis.

3.3 Synopsis Utility Model

Internally, ScroogeDB selects synopses to generate based on a synopsis utility model. Based on the query history and the current set of stored synopses, the utility model calculates an estimated benefit of constructing a new synopsis. For each possible synopsis, we weigh two factors: the probability that this synopsis will be used in the future and the monetary savings obtained when using it.

To predict likely future queries, ScroogeDB determines a probability distribution over queries in the current query workload. Based on past query executions, we calculate the probability of a certain column appearing in a specific part of a query. These probabilities together form the query distribution for the ongoing analysis session. Based on that, we obtain the probability of a synopsis being used in the future. We know that a synopsis can answer a query only when the synopsis supports all columns of the query. Thus, the probability that a synopsis supports a future query is the probability that no unsupported column appears in the query. In addition, the set of synopses we already have influences our utility estimates about others. If a query can be answered by an existing synopsis, there is little benefit in creating a new synopsis for that particular query. Thus, we focus on creating a synopsis that can answer currently unsupported queries.

The expected monetary saving per synopsis usage can be computed based on the size difference between the synopsis and the base tables. We approximate this difference using the ratio of the number of rows in the synopsis to the number of rows in the fact table. Since it is hard to compute the exact number of rows in a synopsis without actually generating it, we estimate the number using a space model. For a synopsis, the model considers the number of distinct values

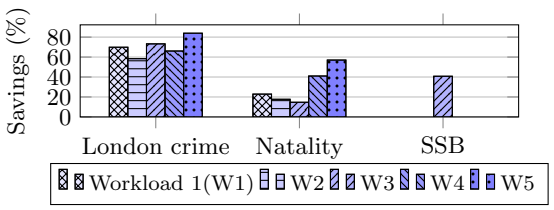

Figure 3: Relative monetary savings per benchmark.

in each of its columns and the total number of rows in the base table.

4. DEMONSTRATION

We present ScroogeDB on top of Google BigQuery[1] and demonstrate its monetary savings compared to standard execution (i.e., executing queries on base tables). We provide a graphical interface which displays deterministic bounds as well as various details regarding the internal decision process (e.g., query rewriting, synopsis utility model, and synopsis generation) of the ScroogeDB system.

4.1 Data Sets and Experimental Results

We use two real-world data sets as well as a standard benchmark in our demonstration. The first data set is about childbirths in the United States between 1969 to 2008 (natality data set)[2]. The second data set contains information about the number of reported crimes per crime type in London (London crime data set)[3]. Also, we use the Star Schema Benchmark (SSB) [5] with a scaling factor of 10. We conducted a user study with five participants to collect analytical query workloads on the two real-world data sets. Giving the participants 20 minutes per data set, they were asked to find interesting facts or trends from the data set. As a result, we collected five query workloads per data set where each workload has up to 76 queries. Since SSB only consists of 13 queries (which is a small number for a query workload), we create ten variants per query by randomly changing the constant values in predicates. Figure 3 illustrates the experimental results on the three benchmarks. We assign a value of 0.9 as the target precision for the experiment. That is, the ratio of the lower bound to the upper bound is at least bigger than or equal to 0.9. Our system was evaluated against a standard execution (i.e., without ScroogeDB) to get initial experimental results. ScroogeDB gives us average monetary savings of 47.23% for all benchmarks and up to 83.39% in the best case for a query workload. By averaging over all five workloads, we get savings of 70.27% for the London crime data set and 30.65% for the natality data set. For SSB, where there is only one workload, we get savings of 40.77%.

4.2 Demonstration Plan

Figure 4 shows a screenshot of the ScroogeDB graphical interface. It allows users to easily formulate aggregation queries on the underlying data sets. The interface visualizes deterministic bounds using black bars (on the right side of the figure). It also displays information regarding the internal decisions of ScroogeDB, including which synopses have been generated and which synopsis has been used per query (under Log in the top-left corner). In addition, monetary fees charged per query execution will be shown to users.

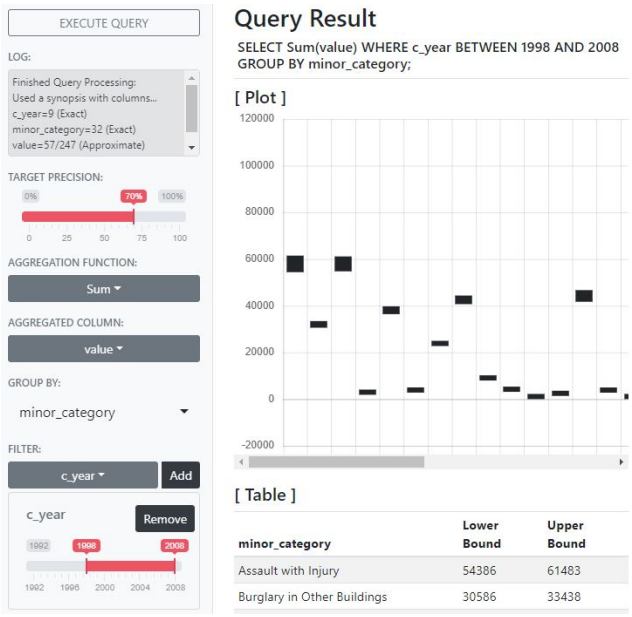

Figure 4: Screenshot of the ScroogeDB interface.

In our demonstration, we present the graphical user interface as a Web application. We provide the option for users to change between ScroogeDB and the standard execution of Google BigQuery. By switching between the two execution methods, users can experience accumulated monetary savings offered by ScroogeDB. Also, users can change the precision constraint for ScroogeDB and observe how it affects the size of the generated synopses. By using smaller synopses, users can save even more monetary fees per query execution.

[1] https://cloud.google.com/bigquery
[2] https://www.kaggle.com/bigquery/samples
[3] https://www.kaggle.com/LondonDataStore/london-crime

5. REFERENCES

[1] 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.

[2] E. Boursier, J. J. Brito, C. Lin, and Y. Papakonstantinou. Plato: Approximate Analytics over Compressed Time Series with Tight Deterministic Error Guarantees. *CoRR*, abs/1808.04876, 2018.

[3] S. Jo and I. Trummer. BitGourmet: Deterministic Approximation via Optimized Bit Selection. In *CIDR*, 2020.

[4] K. Li, Y. Zhang, G. Li, W. Tao, and Y. Yan. Bounded Approximate Query Processing. *TKDE*, 31(12):2262–2276, 2019.

[5] P. O. Neil, B. O. Neil, and X. Chen. Star Schema Benchmark. 2009.

[6] N. Potti and J. M. Patel. DAQ: A New Paradigm for Approximate Query Processing. *PVLDB*, 8(9):898–909, 2015.