Approximate query processing in a data warehouse using random sampling

Trong Duc Nguyen
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/etd

Recommended Citation
Nguyen, Trong Duc, "Approximate query processing in a data warehouse using random sampling" (2020). Graduate Theses and Dissertations. 18195.
https://lib.dr.iastate.edu/etd/18195

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Approximate query processing in a data warehouse using random sampling

by

Trong Duc Nguyen

A Thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

Major: Computer Engineering (Software Systems)

Program of Study Committee:
Srikanta Tirthapura, Major Professor
Suraj Kothari
Goce Trajcevski
Lizhi Wang
Mai Zheng

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation. The Graduate College will ensure this dissertation is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University
Ames, Iowa
2020

Copyright © Trong Duc Nguyen, 2020. All rights reserved.
# TABLE OF CONTENTS

**LIST OF TABLES** ................................................................. v

**LIST OF FIGURES** ............................................................. vi

**ACKNOWLEDGMENTS** ........................................................... ix

**ABSTRACT** ............................................................................ x

**CHAPTER 1. OVERVIEW** ........................................................... 1

1.1 Introduction ......................................................................... 1

1.2 Approximate Query Processing .............................................. 3

1.3 Query-based Sampling Algorithms .......................................... 5

1.4 Pipeline of Queries in the Data Warehouse ............................... 6

1.4.1 Approximate Query Processing with Pipeline ....................... 8

1.4.2 Confidence Interval ............................................................ 9

**CHAPTER 2. PROBLEM STATEMENT** .......................................... 11

2.1 Approach ............................................................................. 14

2.2 Contributions ...................................................................... 14

**CHAPTER 3. REVIEW OF LITERATURE** ....................................... 16

3.1 Approximate Query Processing .............................................. 16

3.2 Random Sampling ................................................................. 17

3.3 Approximate Pipeline Processing in Data Warehouse ................. 20

**CHAPTER 4. OPTIMIZED SAMPLE SPACE ALLOCATION FOR POPULATION ESTIMATE** ................................. 22

4.1 Stratified Random Sampling ................................................ 23

4.2 Overview ............................................................................ 25

4.2.1 Preliminaries .................................................................... 25

4.2.2 Solution Overview ............................................................ 27

4.3 Variance-Optimal Sample Size Reduction .............................. 28

4.3.1 Special Case: Reduction by One Element ............................ 29

4.3.2 Reduction by $\beta \geq 1$ Elements ........................................ 30

4.4 **VOILA**: Variance-Optimal Offline SRS ............................... 36

4.5 Streaming SRS ..................................................................... 37

4.5.1 A Lower Bound for Streaming SRS .................................... 38

4.5.2 **S-VOILA**: Practical Streaming SRS ................................. 41
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6</td>
<td>Experimental Evaluation</td>
<td>45</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Sampling Methods</td>
<td>45</td>
</tr>
<tr>
<td>4.6.2</td>
<td>Data</td>
<td>46</td>
</tr>
<tr>
<td>4.6.3</td>
<td>Allocations to Different Strata</td>
<td>49</td>
</tr>
<tr>
<td>4.6.4</td>
<td>Comparison of Variance</td>
<td>50</td>
</tr>
<tr>
<td>4.6.5</td>
<td>Query Performance</td>
<td>55</td>
</tr>
<tr>
<td>4.6.6</td>
<td>Adapting to a Change in Data Distribution</td>
<td>57</td>
</tr>
<tr>
<td>4.7</td>
<td>Conclusions</td>
<td>59</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 5</th>
<th>Optimized Sample Space Allocation for Group-By Queries</th>
<th>61</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>61</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Contributions</td>
<td>64</td>
</tr>
<tr>
<td>5.2</td>
<td>Preliminaries</td>
<td>65</td>
</tr>
<tr>
<td>5.3</td>
<td>Single Group-by</td>
<td>66</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Single Aggregate, Single Group-By</td>
<td>66</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Multiple Aggregates, Single Group-by</td>
<td>69</td>
</tr>
<tr>
<td>5.4</td>
<td>Multiple Group-Bys</td>
<td>71</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Single Aggregate, Multiple Group-By</td>
<td>71</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Multiple Aggregates, Multiple Group-Bys</td>
<td>75</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Using A Query Workload</td>
<td>77</td>
</tr>
<tr>
<td>5.5</td>
<td>CVOPT-INF and Extensions</td>
<td>79</td>
</tr>
<tr>
<td>5.6</td>
<td>Experimental Evaluation</td>
<td>81</td>
</tr>
<tr>
<td>5.6.1</td>
<td>Accuracy of Approximate Query Processing</td>
<td>82</td>
</tr>
<tr>
<td>5.6.2</td>
<td>Weighted Aggregates</td>
<td>86</td>
</tr>
<tr>
<td>5.6.3</td>
<td>Sample's Usability</td>
<td>86</td>
</tr>
<tr>
<td>5.6.4</td>
<td>Multiple Group-by Query</td>
<td>89</td>
</tr>
<tr>
<td>5.6.5</td>
<td>CPU Time</td>
<td>90</td>
</tr>
<tr>
<td>5.6.6</td>
<td>Experiments with CVOPT-INF</td>
<td>91</td>
</tr>
<tr>
<td>5.7</td>
<td>Conclusion</td>
<td>92</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 6</th>
<th>Sampling over Query Pipeline in Data Warehouse</th>
<th>93</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>The Pipeline of Queries</td>
<td>94</td>
</tr>
<tr>
<td>6.2</td>
<td>Approximate Query Processing on Data Pipelines</td>
<td>96</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Join Pattern</td>
<td>97</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Divide Pattern</td>
<td>98</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Single-chained Data Pipeline</td>
<td>98</td>
</tr>
<tr>
<td>6.3</td>
<td>Computation of the Unbiased Estimations and Confidence Interval</td>
<td>100</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Reducing the Computation of Confidence Interval to the Variance</td>
<td>100</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Computation of the Variance for Terminal Expression</td>
<td>103</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Computation of the Variance for Nonterminal Expression</td>
<td>105</td>
</tr>
<tr>
<td>6.4</td>
<td>Two Passes Model for Pipeline Confidence Interval Computation</td>
<td>109</td>
</tr>
<tr>
<td>6.5</td>
<td>Empirical Study</td>
<td>110</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Data Pipeline on the OpenAQ Dataset</td>
<td>111</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Data Pipeline on a Private Enterprise Dataset</td>
<td>113</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Accuracy of the Approximate Results</td>
<td>114</td>
</tr>
</tbody>
</table>
6.5.4 CI of the Estimations .................................................. 117
6.5.5 Accuracy and Confidence Level of the CI .......................... 117

CHAPTER 7. CONCLUSION .................................................. 120

BIBLIOGRAPHY ............................................................... 121
LIST OF TABLES

Table 5.1  An example Student table ........................................ 77
Table 5.2  An example query workload on the Student table ............. 78
Table 5.3  Aggregation groups and their frequencies produced from the example workload (Table 5.2) .................................................. 78
Table 5.4  Percentage average error for different queries, OpenAQ and Bikes datasets, with 1% and 5% samples, respectively. .................... 85
Table 5.5  Average error of multiple queries answered by one materialized sample, showing the reusability of the sample. ....................... 89
Table 5.6  The sum of the CPU time (in seconds) at all nodes for sample precomputing and query processing with 1% sample for query AQ1 over OpenAQ and OpenAQ-25x. ........................................ 91
Table 6.1  The typical confidence levels and the corresponding value of $z_l$, the quantile of the standard normal distribution. By changing parameter $z_l$, we can set the confidence level $l$. ..................................................... 102
Table 6.2  Summary of the unbiased estimation computations. Details are in sections 6.3.2 and 6.3.3. ............................................................. 102
Table 6.3  The summary of the variance computations. Details are in sections 6.3.2 and 6.3.3. ............................................................. 103
Table 6.4  The CI value and its coverage - percentage of actual results fall within the CI, of Query 3 and 5, OpenAQ pipeline, and some aggregates in PrivEnt pipeline, at different confidence levels. ..................... 118
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>A case study of a pipeline with 5 queries.</td>
<td>6</td>
</tr>
<tr>
<td>4.1</td>
<td>Relative frequencies of different strata. The x-axis is the fraction of points observed so far. At different points in time, the relative (cumulative) frequency of each stratum is shown.</td>
<td>46</td>
</tr>
<tr>
<td>4.2</td>
<td>Relative standard deviations of different strata, demonstrated by normalized cumulative standard deviations observed by the end of each month.</td>
<td>47</td>
</tr>
<tr>
<td>4.3</td>
<td>The number of strata seen so far, and the number of records in data, as a function of time.</td>
<td>47</td>
</tr>
<tr>
<td>4.4</td>
<td>Relative frequencies of different strata in the synthetic dataset change over time.</td>
<td>48</td>
</tr>
<tr>
<td>4.5</td>
<td>Relative standard deviations of different strata in the synthetic dataset change over time.</td>
<td>48</td>
</tr>
<tr>
<td>4.6</td>
<td>Allocation of sample sizes among strata after 9 months, OpenAQ data.</td>
<td>49</td>
</tr>
<tr>
<td>4.7</td>
<td>Allocation due to VOILA changes in allocation over time, OpenAQ data.</td>
<td>50</td>
</tr>
<tr>
<td>4.8</td>
<td>Allocation due to S-VOILA, with Single Element Processing, changes in allocation over time, OpenAQ data.</td>
<td>50</td>
</tr>
<tr>
<td>4.9</td>
<td>Allocation due to S-VOILA with minibatch processing, changes in allocation over time, OpenAQ data. Batch size is one-day of data.</td>
<td>51</td>
</tr>
<tr>
<td>4.10</td>
<td>Cosine distance between the allocations due to VOILA, S-VOILA with single element processing, and S-VOILA with minibatch processing, OpenAQ data.</td>
<td>51</td>
</tr>
<tr>
<td>4.11</td>
<td>Variance of VOILA compares to Neyman and Neyman+ with equal sample size: 1M records, OpenAQ data.</td>
<td>52</td>
</tr>
<tr>
<td>4.12</td>
<td>Variance of streaming S-VOILA, with Single and Minibatch Processing, compared with offline VOILA. Sample size is set to 1M records, for each method, OpenAQ data.</td>
<td>52</td>
</tr>
</tbody>
</table>
Figure 4.13 Relative difference of the variance of S-VOILA, with Single and Minibatch Processing, compared with the optimal variance due to VOILA, OpenAQ data. 53
Figure 4.14 Impact of Sample Size on Variance, in September, OpenAQ data. 54
Figure 4.15 Impact of Batch Size on Variance, OpenAQ data. 54
Figure 4.16 Query Performance as data size varies, with sample size fixed at 100,000. OpenAQ data. 55
Figure 4.17 Query Performance as sample size varies, with data size fixed at 21 million. OpenAQ data. 56
Figure 4.18 Allocation due to VOILA across different strata changes over time, synthetic dataset. 57
Figure 4.19 Allocation due to S-VOILA with Single Element processing changes over time, synthetic dataset. 57
Figure 4.20 Allocation due to S-VOILA with Minibatch processing changes over time, synthetic dataset. Batch size is 100. 58
Figure 4.21 The variance changes due to a sole change in synthetic data. 58
Figure 4.22 Query Performance on synthetic data as size of streaming data increases, with sample size fixed at 1,000 and one stratum’s distribution changed at 10,000. 59
Figure 5.1 Maximum error for MASG query AQ1 and SASG query AQ3 using a 1% sample. 83
Figure 5.2 Average errors, 1% CVOPT sample answers query AQ2 with weight settings. 85
Figure 5.3 Average errors, 5% CVOPT sample answers query B1 with weight settings. 86
Figure 5.4 Sensitivity of maximum error to sample size. MASG query AQ2 answered by samples with various sample rates. 87
Figure 5.5 Sensitivity of maximum error to sample size. SASG query B2 answered by samples with various sample rates. 87
Figure 5.6 Maximum error of SASG queries with different predicates on OpenAQ, answered by one materialized sample, showing the reusability of the sample. 88
Figure 5.7 Maximum error of SASG queries with different predicates on Bikes, answered by one materialized sample, showing the reusability of the sample. 88
Figure 5.8  Maximum error of CUBE group-by queries. .......................... 91
Figure 5.9  Comparison accuracy from CVOPT and CVOPT-INF for SAS queries AQ3 and B2. ................................................................. 92
Figure 6.1  Pipeline DAG for OpenAQ use case has one input table and 5 queries. .... 95
Figure 6.3  Join and Divide patterns, where different single-chained paths intersect. .. 97
Figure 6.4  The DAG for PrivEnt data pipeline. The execution order is from left to right. 113
Figure 6.5  The distribution of the error of different approximate results on 5% sampled PrivEnt pipeline. The median error is less than 10% and the third quantile of most of the aggregates are less than 20%. ......................... 114
Figure 6.6  The median error of the approximate results of different aggregates in PrivEnt pipeline, at various sample rates. ................................. 114
Figure 6.7  The median error of the approximate results of Query 3 and Query 5, OpenAQ pipeline, at various sample rates. ................................. 115
Figure 6.8  Percentage of missing results in Query 3, OpenAQ pipeline and the final table of the PrivEnt pipeline, at various sample rates, comparing to the full results. 116
Figure 6.9  The exact and estimated results, with the 95% CI boundaries, of the estimation in Query 3. Limited to top 50 results for the display purpose. .... 116
Figure 6.10 The exact and estimated results, with 95% CI boundaries, of the PrivEnt pipeline. We hid the labels of the data. Limited to top 10,000 results for the display purpose. ............................................................. 117
Figure 6.11 CI coverage in Query 3, OpenAQ pipeline – percentage of true results fall within the CIs boundaries. The coverages of computed CIs, at various quantile value \( z_t \), are close to the expectation numbers. ....................... 118
I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Srikanta Tirthapura for his guidance, patience, and support throughout this research and the writing of this thesis. His insights and words of encouragement have often inspired me and renewed my hopes for completing my graduate education. I would like to thank my colleagues and collaborators: Bojian Xu, Danny Ming-Hung Shih, Divesh Srivastava, Gabriela Jacques da Silva, Italo Lima, and Michael Levin for their enormous supports. It has been my privilege to work with each and every one of them. I would also like to express my thanks to my committee members for their efforts and contributions to this work: Dr. Suraj Kothari, Dr. Goce Trajcevski, Dr. Lizhi Wang and Dr. Mai Zheng. I would additionally like to thank Dr. Tien Nguyen for his guidance throughout the initial stages of my graduate career.
ABSTRACT

Data analysis consumes a large volume of data on a routine basis. With the fast increase in both the volume of the data and the complexity of the analytic tasks, data processing becomes more complicated and expensive. The cost efficiency is a key factor in the design and deployment of data warehouse systems. Approximate query processing is a well-known approach to handle massive data among different methods to make big data processing more efficient, in which a small sample is used to answer the query. For many applications, a small error is justifiable for the saving of resources consumed to answer the query, as well as reducing the latency.

We focus on the approximate query processing using random sampling in a data warehouse system, including algorithms to draw samples, methods to maintain sample quality, and effective usages of the sample for approximately answering different classes of queries. First, we study different methods of sampling, focusing on stratified sampling that is optimized for population aggregate query. Next as the query involves, we propose sampling algorithms for group-by aggregate queries. Finally, we introduce the sampling over the pipeline model of queries processing, where multiple queries and tables are involved in order to accomplish complicated tasks. Modern big data analyses routinely involve complex pipelines in which multiple tasks are choreographed to execute queries over their inputs and write the results into their outputs (which, in turn, may be used as inputs for other tasks) in a synchronized dance of gradual data refinement until the final insight is calculated. In a pipeline, approximate results are fed into downstream queries, unlike in a single query. Thus, we see both aggregate computations from sampled input and approximate input.

We propose a sampling-based approximate pipeline processing algorithm that uses unbiased estimation and calculates the confidence interval for produced approximate results. The key insight of the algorithm calls for enriching the output of queries with additional information. This enables the algorithm to piggyback on the modular structure of the pipeline without having to perform any
global rewrites, *i.e.*, no extra query or table is added into the pipeline. Compared to the bootstrap method, the approach described in this paper provides the confidence interval while computing aggregation estimates only once and avoids the need for maintaining intermediary aggregation distributions.

Our empirical study on public and private datasets shows that our sampling algorithm can have significantly (1.4 to 50.0 times) smaller variance, compared to the Neyman algorithm, for optimal sample for population aggregate queries. Our experimental results for group-by queries show that our sample algorithm outperforms the current state-of-the-art on sample quality and estimation accuracy. The optimal sample yields relative errors that are $5\times$ smaller than competing approaches, under the same budget. The experiments for approximate pipeline processing show the high accuracy of the computed estimation, with an average error as low as 2%, using only a 1% sample. It also shows the usefulness of the confidence interval. At the confidence level of 95%, the computed CI is as tight as $\pm 8\%$, while the actual values fall within the CI boundary from 70.49% to 95.15% of times.
CHAPTER 1. OVERVIEW

1.1 Introduction

A data warehouse system handles a large volume of data on a routine basis to process various data consumption applications. Over time, we see the humongous increases in both the scale of the data and the complexity of the analytic tasks. The data processing becomes more complicated and thus more expensive. The cost efficiency is a key factor in the design and deployment of data warehouse systems.

Approximate query processing is a well-known approach to handle massive data Acharya et al. (1999a); Agarwal et al. (2014); Babcock et al. (2003); Cao and Fan (2017); Chakrabarti et al. (2001); Chaudhuri et al. (2001a), among different methods to make big data processing more efficient, such as indexing Ding et al. (2016); Subotic et al. (2018); Wang et al. (2015a), caching Adali et al. (1996); Zhao et al. (2014), sketching Anderson et al. (2017); Basat et al. (2018), view materialization Hanson (1987); Uchiyama et al. (1999), etc. Random sampling has been widely used in approximate query processing on large databases Chaudhuri et al. (2017); Chen and Yi (2017); Thompson (2012); Wu et al. (2016); Kandula et al. (2016a); Tillé (2006); Nguyen et al. (2016). Random sampling has the potential to significantly reduce resource usages and response times, at the cost of a small approximation error. In fact, for many applications, the exact answer is unnecessary. Thus a small error is justifiable for the saving of resources consumed to answer the query, as well as reducing the latency. For example, given a query that powers a visualization dashboard where its result is displayed as a figure. The errors in the query answer lead to incorrections in the figure. Most of the time, the small differences in visualization are tolerable. Especially, when the errors are small enough, the difference between approximate and exact figures are virtually unnoticeable. While sampling gives us the flexibility of answering various queries, it does not require so much extra storage space. This feature makes sampling stands out from other approaches to answering big
query efficiently such as caching, pre-computation, or view materialization. While those methods can give the exact answers for the supported set of queries, the size of the additional space needed is proportional to the number of supported queries. When the business needs are involved, sampling becomes a good choice over other approaches, in terms of the ability to work with complex data processing model.

In this work, we focus on the approximate query processing using random sampling in a data warehouse system, including algorithms to draw samples, methods to maintain sample quality, and effective usages of the sample for approximately answering different classes of queries. We study different methods of sampling, focusing on stratified sampling that is optimized for population aggregate query and group-by query. Last but not least, we introduce the sampling over the pipeline model of queries processing, where multiple queries and tables are involved in order to accomplish complicated tasks.

The pipeline model allows the user to break down their analysis into multiple steps, in which each step is relatively smaller. The pipeline is not only helpful in making the analytic task more readable and easier to debug and maintain, but also useful in the modularity of the analysis. With the pipeline model, data flow becomes logical and intermediate results can be reused. The pipeline model has been widely studied and used Jacques-Silva and Zhang (2018); Bowers et al.; Pfleiger et al.; Huang et al..

We will focus on the computation of confidence interval for the estimations when we apply a sample algorithm on a pipeline. Since the results from the sample are approximated, one of the main concerns of sampling’s usefulness is how good or bad the approximations are, i.e., how far way the estimates are from the ground-truth answers. The confidence interval is a good measurement that can be used as enforcement for each estimated result. In traditional sampling models, confidence interval has been widely studied Haas (1997); Cohen (1994); Johns (1988); Schenker (1985); Nedelman et al. (1995). Numerically, it can be present along with the estimate, e.g., the estimate $x = 100 \pm 5$ says the estimated value of $x$ is 100 and, with high probability, the unknown actual value falls within the range $(95 - 105)$. 
As the pipeline model is complex, computing the confidence for the estimation is challenging. The aggregates that are directly on the sampled table have the information about the randomness, but downstream aggregates, which query on top of the approximate tables, do not have this information. The needed information has been propagated along with the main results in the pipeline. Secondly, there are many different computation operators in a pipeline, in which provide confidence interval is not always trivial. Even for those operators that confidence interval is well formulated, each of them requests different intermediate information to have the confidence interval computed. Simply combining all the needed information is infeasible. The number of extra works will blow up the complexity of the pipeline, which is already complicated.

By addressing the challenging problem of computing confidence interval for sampled estimations in the pipeline computational model, we are expected to significantly improve the usefulness of approximate query processing, especially for complex pipelines. The confidence interval not only provides the measurement of estimate’s accuracy but also leads to the tremendous potential of applications, related to using sampling, such as optimizing the sample rate, dynamic sampling, etc..

1.2 Approximate Query Processing

Approximate query processing is essential for handling massive data. Different methods of approximations provide interactive response time when exploring massive datasets, and are also needed to handle high-speed data streams. Some methods are lossless, while others are lossy compaction of the data. But they share the same high-level approach of using synopsis rather than the entire dataset. The synopsis could be composed in different ways, include, but not limited to, random samples Babcock et al. (2003); Ganti et al. (2000); Haas (1997); Brown and Haas (2006), histograms Chaudhuri et al. (1998); Muralikrishna and Dewitt (1988); Ioannidis (1993); Ioannidis and Poosala (1995), wavelets Cormode et al. (2011); Gilbert et al. (2001); Vitter and Wang (1999), sketches Tirthapura et al. (2006); Xu et al. (2008); Cormode et al. (2007); Indyk et al. (2000), materialized views Olken and Rotem (1992); Hanson (1987); Uchiyama et al. (1999), etc.. Each of
such synopsis captures vital properties of the original massive data while typically occupying much less space. For example, given the original data consists of large time serial numbers, we can have a materialized view that pre-computes the number of observations, \( n \); the sum of all the values, \( \sum x \); the sum of the squares of the values, \( \sum x^2 \). This view helps not only with the pre-computed aggregates but also aggregates referred on top of it, such as with the ratio of the sum and the count is the average, \( \frac{\sum x}{n} \), and the variance is a ratio of the sum of squares to the count, less the square of the average, \( \frac{\sum x^2}{n} - \left( \frac{\sum x}{n} \right)^2 \).

In the above example, the aggregates are lossless, however, we may need to know more about the data than merely its variance. The other estimations may not have such properties, or even can’t be composed because the view with three-value summary does not suffice. To answer more various types of queries, an appropriate type of synopses are needed. In general, any type of approximate query processing, no matter how complex they are to compute, will have weak points. Indeed, for many of these questions, no synopsis can provide the exact answer. For example, the row-select query, the query answers collectively describe the data in full, and so any synopsis would effectively have to store the entire dataset. Other cases are when the result of the query is error-sensitivity, \( i.e., \) the query to create bills, fill out tax information, \( etc.\).

Nonetheless, a wide range of classes of the query, where the small error can be tolerated, can be well supported by approximate query processing. In these cases, the requirements are much relaxed. The key objective is not obtaining the exact answer to a query, but rather receiving an accurate estimate of the answer. \( e.g., \) receiving an answer that is within 1% of the true result is adequate for most of the cases. It might suffice to know that the true answer is approximated \( x \approx 1 \times 10^6 \), without knowing that the exact answer is \( x = 1,010,483.6367 \). Thus we can tolerate approximation. Such synopses that provide approximate answers are practically useful in analytic applications that process large datasets.

As approximate query processing is widely applied in practice, the research community has also been studying it thoroughly in multiple aspects such as accuracy, space and time efficiency, optimality, error bounds on query answers, incremental maintenance, \( etc.\).
1.3 Query-based Sampling Algorithms

We design sampling algorithms to serve different classes of queries, starting from the simple population aggregate query, that produces a single aggregation. Next, we design sampling algorithms for the group-by query, in which the answer is a collective aggregate result. Lastly, we target the complex query pipeline where it involves multiple queries.

For the simple query of population aggregate, general unbiased sample, e.g., uniform sample, is straightforward to implement and can be widely used. However, the sample may not well represent the original data. For example, given a simple dataset of 1000 elements, that contains 990 Trues and 10 False values, a uniform sample high likely contains all the True elements. Otherwise, to have False values represented, the uniform sample will be relatively large, which is again the purpose of using the sample. Stratified random sampling is a fundamental biased sampling technique that provides accurate estimates for aggregate queries using a small size sample, and has been used widely for approximate query processing. A key question in stratified sampling is how to partition a target sample size among different strata. While Neyman allocation Neyman (1934) provides a solution that minimizes the variance of an estimate using this sample, it works under the assumption that each stratum is abundant, i.e., has a large number of data points to choose from. This assumption may not hold in general: one or more strata may be bounded, and may not contain a large number of data points, even though the total data size may be large.

We present VOILA, an offline method for allocating sample sizes to strata in a variance-optimal manner, even when one or more strata may be bounded. Our results from experiments on real and synthetic data show that VOILA can have significantly (1.4 to 50.0 times) smaller variance than Neyman allocation.

Next step, we target a more complex query class with a group-by clause. This is a widely used type of query where the query firstly groups data according to one or more attributes, and then aggregate within each group after filtering through a predicate. The challenge with group-by queries is that a sampling method cannot focus on optimizing the quality of a single answer (e.g.
the mean of selected data), but must simultaneously optimize the quality of a set of answers (one per group).

We present \texttt{CVOPT}, a query- and data-driven sampling framework for a set of queries that return multiple answers, \textit{e.g.}, group-by queries. To evaluate the quality of a sample, \texttt{CVOPT} defines a metric based on the norm (\textit{e.g.} $\ell_2$ or $\ell_\infty$) of the coefficients of variation (CVs) of different answers and constructs a stratified sample that provably optimizes the metric. \texttt{CVOPT} can handle group-by queries on data where groups have vastly different statistical characteristics, such as frequencies, means, or variances. \texttt{CVOPT} jointly optimizes for multiple aggregations and multiple group-by clauses and provides a way to prioritize specific groups or aggregates. It can be tuned to cases when partial information about a query workload is known, such as a data warehouse where queries are scheduled to run periodically.

Our experimental results show that \texttt{CVOPT} outperforms the current state-of-the-art on sample quality and estimation accuracy for group-by queries. On a set of queries on two real-world data sets, \texttt{CVOPT} yields relative errors that are $5\times$ smaller than competing approaches, under the same budget.

1.4 Pipeline of Queries in the Data Warehouse

Not only the volume of the data but also the complexity of the analysis tasks is increasing over time. Gaining the non-trivial insights requites complicated explorations and mutations of the
data. Eventually, a single query is not sufficient to efficiently handle complex tasks. To address the problem, the model of the pipeline of queries is introduced Jacques-Silva and Zhang (2018).

A pipeline of queries, or a pipeline – in short, is a collection of queries and tables, placed in a topological order of execution. A pipeline is a directed acyclic graph (DAG), where the tables are represented by nodes and the queries are denoted by edges. The results of a query are materialized as a table, that may serve other queries down the stream. Figure 1.1 shows an example of a pipeline that consumes data from a logging system to power a visualization dashboard. The pipeline contains 4 queries and 6 tables. Instead of a single complicated query, that joins all the source tables to do such massive computation at one, the pipeline breaks the complex task into sub-tasks. It organizes the computation as a DAG of sub-tasks and intermediate results. Each sub-task can be done by a much smaller query. Due to the great benefits that the model provides, it has been practically adopted in industrial, large-scale data warehouse systems Jacques-Silva and Zhang (2018); Bowers et al.; Pfleiger et al.; Huang et al..

The pipeline model brings multiple benefits, especially in a large-scale setting:

- **Modularity:** The complex task is broken down into multiple components, each is well defined and is designed to achieve a single or few intermediate results. Data engineers have better control over their pipeline. Multiple small, easy to understand components are easier to debug, maintain, and evolve the analysis than a huge, super-complicated query.

- **Readability:** Complex task makes the query hard to read to humans, especially with SQL language, since it is not designed to carry the logic as a programming language. Thus, having self-contain components reduces complexity. Each query in the pipeline can be as simple as doing a projection or an aggregation, make them easily understandable.

- **Reusability:** Even with highly optimized query engines, the complex tasks may lead to redundancy in query computation. In the pipeline model, intermediate results are materialized. Thus, they can be reused instead of duplication in the computation. Furthermore,
intermediate results can also be shared among pipelines. In practical deploy, data warehouse systems have tables that served multiple pipelines.

- **Scalability:** The pipeline model allows the system to be scaled in both the magnitude of the data and the number of tasks can be processed. The supporting infrastructure is a combination of multiple workers. Instead of having one worker do a complex task, a pipeline can be processed by multiple workers in parallel.

### 1.4.1 Approximate Query Processing with Pipeline

As large-scale data analysis is expensive, while many tasks do not require the exact answer, approximate query processing is a good approach to reduce resource consumption. Random sampling has been widely used in approximate query processing, especially for data warehouse setting Chaudhuri et al. (2017); Chen and Yi (2017); Thompson (2012); Cochran (1977); Lohr (2009); Tillé (2006). Traditionally, with single query processing, the approximate answer is derived from the sampled table where the randomness can directly be handled. In this work, we present a model of random sampling for the pipeline of queries. As the pipeline is a complex data processing model where not only one query but a collection of queries are involved, the approximate answer of one query becomes the input for another query. Thus the uncertainty is carried out and controlling the randomness becomes much more complicated.

In the pipeline model, when a table is sampled, the part following it contains approximate results. Also, there may be multiple input tables involved. Different choices of samples to be sampled lead to the different schema of applying sampling for a pipeline. We observe that the processing of a query over a sampled table is different than of the query over an approximate table. The sampled table contains a subset of the data, in which each record is an actual observation from the original data. The randomness in estimating from those tables is from whether or not each record is selected into the sample. On the other hand, in the approximate table, the records are estimated. Thus, not only may a subset of the original results are seen, but the observed data itself
contains uncertainty. Therefore, applying sampling over the pipeline query model is challenging and non-trivial.

1.4.2 Confidence Interval

Sampling has been shown to be an efficient approach to handle massive data in the warehouse. By querying to the sample table, which has a subset of the data, we consume fewer resources and the user gets results faster. The trade-off is that answers are not exactly accurate, rather they are approximate answers. While exact answers are not necessary for most of the business needs, users still demand to know how accurate the approximate answers are. Confidence Interval (CI) can enrich the estimation. The CI of an estimation $x$ is a range in such with high probability the true value belongs to:

$$x \in [\bar{x} - \delta; \bar{x} + \delta]$$

In which $\bar{x}$ is the unbiased estimation of $x$, and $\delta$ is the CI of this estimation. The estimation is also written in form of $x \approx \bar{x} \pm \delta$. For example, given a 90% CI of an approximate count query answer is $100 \pm 5$, we can expect that, with 95% certainty, the true count is between 95 and 105. The CI $\delta$ not only tells the user how accuracy each aggregate but also be used in different applications, such as to visualize the error bar and to guide the user on deciding the sample rate, etc..

While building suppuration, both theoretically and systematically, to sampling for a complex pipeline of queries, integrating CI computation as a part of the sampling model is essential. In opposite to computing the CI for a single query, providing CI for a pipeline of queries is non-trivial, due to the complexity of the pipeline as well as the model we have to execute the pipelines. For example, on the pipeline described in Figure 1.1, the computation of a $\text{SUM}$ aggregate in Query 1, that takes sampled table1 as input, is different than a $\text{SUM}$ aggregate of Query 4, that queries to an approximate table5. A pipeline is a complicated DAG of multiple queries, each query may be complicated and has nested queries. Each node in the DAG is processed at a time. Intermediate results are materialized as a table, thus at the final step, where the actual output is computed, tracing back up to the input tables, where we apply sampling is impractical. The intermediate
results are needed to be carried on along the pipeline. Along with the approximate query processing, using sampling, over the pipeline model, we introduce the integrated computation of the CI.
CHAPTER 2. PROBLEM STATEMENT

We first define the problem we aim to solve in this work, followed by the motivation for the research as well as the challenges of solving it.

**Problem statement:** *Design the computational models for approximate query processing, including drawing the query-driven optimal random samples and efficiently using the random sample in a data warehouse setting.*

Approximate query processing is a powerful approach to handling large-scale data sets. Sampling is commonly used in approximate query processing as the sample is smaller, thus it is cheaper to query upon but can well represent the original data. Where the small error is tolerable, approximate query processing is very useful in reducing resource consummations as well as reduce latency.

We proposed sample algorithms to draw the sample that best represents the data. Although a uniform random sample is simple and widely used, it selects records into the sample uniformly at random. The sample does not well represent the original data as underpopulated groups may be missing or underrepresented in the sample. With the same sample space, given information about the original dataset, we can have a better representative by a biased sample that captures the frequency skewness of the data, especially in representing infrequent portions of the data. Moreover, given both the data statistics and the information about the query workload, *i.e.*, the classes of the query, that the data serves, we aim to draw the sample that not only well represents the data but also be optimal to serve the queries.

Although random sampling has been studied along with the history of data processing, drawing and maintaining optimal samples are challenging:

- The definition of an optimal sample is unclear, thus creating a good random sample is nontrivial. A universal optimal sample is unachievable as the sample is only the synopsis constructed...
from a subset of the data, no sample can serve all queries, especially with a selective query that the result depends on a single record. Thus, we first narrow down the target to be the sample is drawn to serve aggregate queries.

- The query can have a group-by clause, in which data is divided into groups. Since data distribution of each group varies significantly, we classify the aggregate queries into 2 subsets: population aggregate and group-by aggregate. In other words, we aim to serve 2 different classes of SQL queries, with and without GROUP BY clause. This strategy helps us address the challenge of defining the measurement of how good a sample is.

- The reusability of the sample is a strong requirement. Creating a sample for each query is expensive and against the purpose of reducing resource consumption. On the other hand, each query is unique due to the power of the query language. The SQL query is very flexible, a query can be a combination of multiple aggregates, with or without predicates, groups by, join, etc.. And a query can have multiple nested subqueries. We aim to create a sample that can be reused for multiple queries.

Furthermore, we apply the sampling in approximate query processing in the pipeline - a complex data warehouse settings. The pipeline model breaks a complex data analysis task into multiple steps, in the form of a DAG of queries and intermediate results, as shown in Figure 1.1. The sampled table not only serves a single query but multiple queries. Moreover, the estimated results are fed into downstream queries. When the uncertainty is amplified after each step, the computation of unbiased estimation in downstream steps is nontrivial. Moreover, providing the confidence interval for the estimation is challenging, due to the lack of direct information of the randomness. Each estimate has its own unique computation, which depends on not only the function but its position in the complex DAG. We aim to provide a systematic computation model that derives both the unbiased estimate and its confidence interval for the approximate pipeline.

We introduce the sampling model that not only delivers the approximate results but also the confidence interval for each estimate. The pair come together in form of $x \approx \bar{x} \pm \delta$, in which $\bar{x}$
and $\delta$ are the unbiased estimation and the CI of the estimation $x$. For example, while the true value of an \texttt{sum} aggregate is $SUM = 10242$ while its estimation could be $SUM \approx 10000 \pm 500$. As the answers are approximated, naturally, providing a good measurement of the accuracy of the estimate becomes essential. The \textit{Confidence Interval} is a good method to do such measurements. The confidence interval provides a measure of accurateness of the estimate as they are needed in any approximate answer. Moreover, the confidence interval also provides a means to strengthen the sampling over a pipeline, in which it can be incorporated in the sample selection procedure and sample rate decision.

The great usefulness of the confidence interval comes with several challenges to compute it.

- Computing confidence interval is non-trivial. The computation is by operators. While some operators, such as \texttt{Sum} and \texttt{Count}, have the confidence interval well defined, others, such as \texttt{Max}, \texttt{Percentile}, etc., are not clear.

- The pipeline of queries is complex. Not as a traditional sampling model where the sample is drawn from a single table, in the pipeline model there are multiple tables involved. As multiple sources of randomness are present, the computation of confidence interval becomes more complicated.

- The non-directed computation in the pipeline. In a simple sampling instance, the sample is drawn from a single table, and estimations are upon such a single sample. On the other hand, a pipeline is a DAG of queries. For maximizing the effect of resource-saving, the samples tend to be in the early stages of the pipeline. Meanwhile, the confidence intervals are needed at the final step of the pipeline. Following the flow of computation, queries are on top of the estimated results. Such indirect computation is challenging.

- Exponential of the computation space. As the covariance of different attributes is needed in the computation, providing all the covariance of all combinations is infeasible.
2.1 Approach

In this work, we firstly study the approximate query processing, using the sample table, on a large-scale. This study builds up the basic statistic and optimization backgrounds for using sampling. Since sampling has been thoroughly studied in the research community as well as widely used in industrial systems and applications, we start by reviewing the literature of using sampling. Next, we focus on studying stratified sampling, an interesting sampling method that advances the sample power to serve approximate query processing better with limited given resources. We propose sampling algorithms that fit the specific purpose of query processing, particularly, to serve the aggregate queries, with and without group-by.

Secondly, we introduce the sampling for the model of the query pipeline. We start with the based model of the pipeline, computation method, and a case study pipeline. We then focus on computation models for supporting sampling on the query pipeline. The main challenge we aim to address is to provide a confidence interval for each estimated result computed in the pipeline. We propose a model to compute the confidence interval for a set of commonly used aggregate functions. Not only address the problem of computing confidence interval, but we also show the usefulness of the confidence interval in approximate query processing. The confidence interval is not only enriching the estimated results but also provides a means to optimize the sampling model. For example, one research question that we aim to address is using the confidence interval to allocate the sample space. Optimizing sample space by the characteristic of the data Nguyen et al. (2019); Cormode et al. (2011); Ding et al. (2016) has been well studied. With confidence interval, we can optimize sample size based on user-defined error bound as well.

2.2 Contributions

This research aims at contributing a novel model of drawing and using random sampling in approximate query processing.
1. We study the problem of approximate query processing in a massive data warehouse. By using different methods of sampling, including uniform and stratified sampling, we show to the potential of reducing resource consumption and shorten the query latency in processing large datasets.

2. We proposed query-driven random sampling algorithms, that create and maintain the optimal samples based on statistics of the dataset and the knowledge of the query workload, in both streaming and static data management manner.

3. We introduce the approximate query processing for the query pipeline model. While the pipeline model allows a complicated task to be broken down into small, manageable sub-tasks, applying sampling on top of it further reduce the cost of processing complex task.

4. We design the method to compute both the unbiased estimation and the confidence interval for each estimation in approximate pipeline processing. The method support projection, basic algebra operators, and common aggregate COUNT, SUM, and AVG. Join is partially supported under a certain assumption that guarantees the equivalent of sampling apply before and after joining.
CHAPTER 3. REVIEW OF LITERATURE

In this chapter, we study the background knowledge from previous works, including sampling techniques and approximate query processing, over static or streaming data. We present the state-of-art technologies and we also discuss what contribution we could make in this research field. We study the background knowledge in the computational model of the query pipeline.

3.1 Approximate Query Processing

Approximate query processing methods are necessary for dealing with massive data and has been extensively studied Chaudhuri et al. (2017); Babcock et al. (2003); Agarwal et al. (2014); Ganti et al. (2000); Peng et al. (2018); Flajolet (1985); Gilbert et al. (2001); Goldreich (1997); Brown and Haas (2006). They are good means of reducing interactive response times and resource consummations when exploring massive datasets. These common methods proceed by computing a lossy, compact synopsis of the data, and then executing the query of interest against the synopsis rather than the entire dataset.

Histograms have been well studied Gibbons et al. (1995); Ioannidis and Christodoulakis (1993); Gilbert et al. (2002) and have been incorporated into the query optimizers of virtually all commercial relational databases. It summarizes a dataset by grouping the data values into subsets, or “buckets”, and then, for each bucket, computing a small set of summary statistics that can be used to approximately reconstruct the data in the bucket.

Sketch summaries are particularly well suited to streaming data Tirthapura et al. (2006); Xu et al. (2008); Cormode et al. (2007); Indyk et al. (2000). Linear sketches, for example, view a numerical dataset as a vector or matrix and multiply the data by a fixed matrix. Such sketches are massively parallelizable. They can accommodate streams of transactions in which data is both
inserted and removed. Sketches have also been used successfully to estimate the answer to COUNT DISTINCT queries, a notoriously hard problem.

Histograms and sketch share a drawback of limited usage. These synopses are highly compact and particularly good for the set of designed queries, but they are lack of the generality to handle ad-hoc queries. Sampling is a relatively more expensive synopsis, that better captures the baseline data. Sampling has been the most widely used in approximate query processing on both static Chaudhuri et al. (2017); Chen and Yi (2017); Thompson (2012); Cochran (1977); Lohr (2009); Tillé (2006) and streaming data Nguyen et al. (2019); Hentschel et al. (2018); Haas (2016); Zhang et al. (2016); Ahmed et al. (2017). The reservoir sampling Mcleod and Bellhouse (1983); Vitter (1983) method for maintaining a uniform random sample on a stream has been known for decades, and many variants have been considered, such as weight-based sampling Efraimidis and Spirakis (2006); Braverman et al. (2015), stream sampling under insertion and deletion of elements Gemulla et al. (2008), distinct sampling Gibbons and Tirthapura (2001), sampling from a sliding window Babcock et al. (2002b); Gemulla and Lehner (2008); Braverman et al. (2009), and time-decayed sampling Cormode et al. (2009b,a). We discuss random sampling in the next section.

A majority of prior work on using stratified sampling in approximate query processing Acharya et al. (1999a, 2000); Chaudhuri et al. (2007); Joshi and Jermaine (2008); Agarwal et al. (2013) has assumed static data. With the emergence of data stream processing systems Babcock et al. (2002a) and data stream warehousing systems Johnson and Shkapenyuk (2015), it is important to devise methods for streaming stratified sampling with quality guarantees.

### 3.2 Random Sampling

Over the decades, many researchers have been studying different sampling techniques based on statistics to improve the accuracy of the collected sample Cochran (1977); Lohr (2009); Mcleod and Bellhouse (1983); Thompson (2012); Tillé (2006). In the following paragraphs, we will briefly introduce the background knowledge of some sampling techniques, including uniform sampling and stratified sampling.
Uniform random sampling also known as *simple random sampling*, is a method to draw a random sample in which the element is selected uniformly at random Cormode et al. (2011); Lohr (2009); Tillé (2006). In other words, each individual in the population is chosen with the same probability. For example, if we want to sample \( m \) elements from a population of \( n \) elements \( (m < n) \), each element in this population has equal probability of \( \frac{m}{n} \) to be chosen into the sample set.

Reservoir sampling Vitter (1985, 1983); Efraimidis and Spirakis (2006) is an extended random sampling technique for streaming data, that guarantees equal probability of selecting each element in a population without prior knowledge of population size. Reservoir sampling is commonly known for its usefulness in data streaming where total length is unknown.

Stratified sampling is a biased sampling, where the population is partitioned into subgroups called “strata”. From within each stratum, uniform random sampling is used to select a per-stratum sample. All per-stratum samples are combined to derive the “stratified random sample”. Stratified random sample in the online setting Thompson (2012) can be viewed as a type of weight-based reservoir sampling where the weight of each stream element is changing dynamically, based on the statistics of the stratum the element belongs to. Since the weight of each stream element changes dynamically, even after it has been observed, prior work on weighted reservoir sampling Efraimidis and Spirakis (2006) does not apply here, since it assumes that the weight of an element is known at the time of observation and does not change henceforth. Meng Meng (2013) considered streaming stratified sampling using population-based allocation. Al-Kateb *et al.* Al-Kateb and Lee (2010, 2014) considered streaming stratified sampling using power allocation, based on their prior work on the adaptive reservoir sampling Al-Kateb *et al.* (2007). Lang *et al.* Lang et al. (2016) consider machine learning methods for determining the per-item probability of inclusion in a sample. This work is meant for static data and can be viewed as a version of weighted random sampling where the weights are learned using a query workload.

Specified samples, that is optimized for the given query workload or query pattern, is well studied. Ganti *et al.* Ganti et al. (2000) addresses low selectivity queries using workload-based sampling, such that a group with a low selectivity is sampled as long as the workload includes
queries involving that group. Different techniques have been used in combination with samplings, such as indexing Ding et al. (2016); Wang et al. (2015b); Chaudhuri et al. (2001b), or aggregate precomputation Peng et al. (2018).

Chaudhuri et al. (2007) formulate approximate query processing as an optimization problem. Their goal is to minimize the $\ell_2$ norm of the relative errors of all queries in a given workload. Their approach to group-by queries is to treat every group derived from every group-by query in the workload as a separate query. In doing so, their technique does not handle overlap between samples suited to different groups. In contrast, our framework considers the overlaps and interconnections between different group-by queries in its optimization.

Kandula et al. (2016b) consider queries that require multiple passes through the data and use random sampling in the first pass to speed up subsequent query processing. This work can be viewed as query-time sampling while ours considers sampling before the full query is seen. Further, Kandula et al. (2016b) does not provide error guarantees for group-by queries, as we consider here. A recent work Nguyen et al. (2019) has considered stratified sampling on streaming and stored data, addressing the case of full-table queries using an optimization framework. This work does not apply to group-by queries.

Rösch and Lehner (2009) propose non-uniform sampling to support group-by queries, where different groups can have vastly different variances. This proposed algorithm is a heuristic without a guarantee on the error, and in fact, does not have a well-defined optimization target. In our work, we will study a sample algorithm that has provable guarantees holding even in case of multiple aggregations and/or multiple group-bys.

“Congressional sampling” Acharya et al. (2000) targets sampling for a collection of group-by queries, especially those that consider a “cube by” query on a set of group-by attributes. Congressional sampling is based on a hybrid of frequency-based allocation (the “house”) and fixed allocation (the “senate”). However, congressional sampling ignores the coefficients of variation (and hence also the variances) of different strata in deciding allocations, it only uses the frequency. That results in a non-optimal allocation.
A recent work “Sample+Seek” Ding et al. (2016) uses a combination of measure-biased sampling and an index to help with low-selectivity predicates. Measure-biased sampling favors rows with larger values along with the aggregation attribute. This does not consider the variability within a group in the sampling step – a group with many rows, each with the same large aggregation value, is still assigned a large number of samples. Thus, it does not favor groups with larger CVs. Note that those groups with larger variations in the aggregation attribute. Further, “Sample+Seek” does not provide a sampling strategy that is based on an optimization framework.

### 3.3 Approximate Pipeline Processing in Data Warehouse

Approximate query processing is a well-known approach to conduct analysis over massive data Acharya et al. (1999a); Agarwal et al. (2014); Babcock et al. (2003); Cao and Fan (2017); Chakrabarti et al. (2001); Chaudhuri et al. (2001a). Among different synopses, such as indexing, caching, and sketching, random sampling has been widely used to approximate results of queries on large databases Chaudhuri et al. (2017); Chen and Yi (2017); Wu et al. (2016); Kandula et al. (2016a); Nguyen et al. (2016).

The data pipeline model is commonly used on the analysis of large datasets as it enables us to break down complex tasks into a flow of manageable sub-tasks. Applying AQP on top of the data pipeline model potentially reduces the resources needed for processing massive datasets while still resulting in good approximations. This is the case, especially when considering that samples of massive datasets are still pretty large, resulting in a good representation of the original dataset. To the best of our knowledge, this work is the first to describe the application of sampling on a data pipeline and detailing how we can do the computation of confidence intervals on such scenarios.

For a single-step AQP, the statistical inequalities and the central limit theorem has been used to compute the CI of the approximate result Chaudhuri et al. (2007); Hu et al. (2009); Wu et al. (2010). Pansare et al. Jermaine et al. (2007) developed a Bayesian framework to infer the CI of the approximate answers. However, this approach is limited to simple group-by aggregate queries. Other works have focused on specific types of queries. Charikar et al. Charikar et al. (2000)
studied distinct value estimation from a sample; Jermaine et al. Jermaine et al. (2005) proposed an algorithm to quantify aggregate queries with subset testing. These works neither provide a systematic way of computing the confidence interval nor are able to handle the query pipeline scenario, where the randomness information, for the most part, is not directly accessible.

An alternative approach to compute the confidence interval for approximations is bootstrap, a method adopted from statistics studies Bickel et al. (1981); Efron and Tibshirani (1994); Kleiner et al. (2013). In the database community, recent works Agarwal et al. (2014); Zeng et al. (2014); Pol and Jermaine (2005) have used bootstrap to quantify the quality of approximate query answers. Although bootstrap can support a wider range of aggregation queries, it is known to be costly, due to its resample procedure and multiple tries in order to have reliable results. Pol et al. Pol and Jermaine (2005) focus on efficiently generating bootstrap resamples in a relational database setting, while Laptev et al. Laptev et al. (2012) target MapReduce platforms and study how to overlap computation across different bootstrap resamples. Nonetheless, the computational cost of bootstrap is far more expensive than those of the original sampled query, which is directly against the purpose of approximate query processing. Especially in complex query model, such complexity is amplified as the pipeline contains multiple queries.
CHAPTER 4. OPTIMIZED SAMPLE SPACE ALLOCATION FOR POPULATION ESTIMATE

In this chapter, we discuss stratified random sampling (SRS) and optimizing the sample space for SRS to serve population estimates. First, we present an offline algorithm for variance-optimal SRS for data that may have bounded strata. Our algorithm VOILA (Variance Optimal Allocation) computes an allocation that has provably optimal variance among all possible allocations of sample sizes to different strata. While prior work assumes that there are no strata with small volumes of data, which is often violated in real data sets, our analysis makes no such assumptions. VOILA is a generalization of Neyman allocation and reduces to Neyman allocation in the case when every stratum is abundant.

We present a lower bound showing that any streaming algorithm for SRS that uses a memory of $M$ records must have, in the worst case, a variance that is a factor of $\Omega(r)$ away from the variance of the optimal offline algorithm for SRS that uses a memory of $M$ records. This lower bound is tight since there exist streaming algorithms for SRS whose variance matches this bound in the worst case.

We present S-VOILA, a streaming algorithm for SRS that is locally optimal with respect to variance – upon receiving new elements, it (re-)allocates sample sizes among strata so as to minimize the variance among all possible re-allocations. S-VOILA can be viewed as the online, or dynamic counterpart of the optimization that led to VOILA, which is based on optimizing the variance using a static view of data. S-VOILA can also deal with the case when a minibatch of multiple data items is seen at a time, rather than only a single item at a time – re-allocations made by S-VOILA are locally optimal with respect to the entire minibatch and are of higher quality for larger size minibatches than when a single element is seen at a time. In our experimental study, we found that the variance of S-VOILA is typically close to that of the offline algorithm VOILA, and the variance
of S-VOILA improves as the size of the minibatch increases. Since it can deal with minibatches of varying sizes, it is well-suited to real-world streams that may have bursty arrivals.

The algorithms for offline SRS (VOILA) and streaming SRS (S-VOILA) are both based on a technique for reducing the size of an existing stratified random sample down to the desired target size such that the increase in the variance of the estimator based on the final sample is optimized. This technique for sample size reduction may be of independent interest in other tasks such as sub-sampling from a given stratified random sample.

We present a detailed experimental evaluation using real and synthetic data, considering both the quality of the sample and the accuracy of query answers. Our experiments show that (a) VOILA can have a significantly smaller variance than Neyman allocation, and (b) S-VOILA closely tracks the allocation as well as the variance of the optimal offline algorithm VOILA. As the size of the minibatch increases, the variance of the samples produced by S-VOILA decreases. A minibatch of size 100 provides most of the benefits of VOILA, in our experiments on real-world data.

4.1 Stratified Random Sampling

The simplest method for random sampling is uniform random sampling, where each element from the entire data (the “population”) is chosen with the same probability. But uniform random sampling may lead to a high variance in estimates for aggregate queries. For instance, consider a population \( D = \{1, 1000, 2, 4, 4, 2, 1050, 1200, 1, 1300\} \), and suppose we wanted to estimate the sum of the population. A uniform random sample of size two will lead to an estimate with a variance of \( 1.3 \times 10^7 \). An alternative sampling method is stratified random sampling (SRS), where the population is partitioned into subgroups called “strata”. Within each stratum, uniform random sampling is used to select a per-stratum sample. The different per-stratum samples are then combined to derive the “stratified random sample”. Suppose that the population is divided into two strata, one with elements \( \{1, 2, 4, 2, 1\} \) and the other with elements \( \{1000, 1050, 1200, 1300\} \). A stratified random sample of size two can choose one element from each stratum, yielding an
estimate with the variance of $2 \times 10^5$, 46 times smaller than what was possible with a uniform random sample of the same size.

In SRS, there is the flexibility to emphasize some strata over others, through controlling the allocation of sample sizes; for instance, a stratum with a high standard deviation of values within can be given a larger allocation than another stratum with a lower standard deviation. In the above example, if we desire a stratified sample of size three, it is beneficial to allocate a smaller sample size of one to the first stratum and a larger sample size of two for the second stratum, since the standard deviation of the second stratum is higher. Doing so, the variance of the estimate of the population sum reduces to approximately $1 \times 10^5$. SRS has been used widely in database systems for approximate query processing Agarwal et al. (2013); Chaudhuri et al. (2007); Joshi and Jermaine (2008); Acharya et al. (1999a, 2000).

Suppose that there are $r$ strata, numbered from 1 to $r$, and that the mean, standard deviation, and the number of items in the $j$th stratum are $\mu_j$, $\sigma_j$, and $n_j$ respectively. Suppose that the target sample size is $M$ (total across all strata). We measure the quality of a stratified random sample through the variance in the estimate of the population mean$^1$, computed using this sample. In “uniform allocation”, each stratum $j$ gets an identical allocation of sample size of $s_j = M/r$. In “proportional allocation”, a stratum is allocated a sample size proportional to the number of elements in it. A commonly used method that is believed to yield the smallest variance for an estimate of a population mean is “Neyman allocation” Neyman (1934); Cochran (1977), where stratum $j$ gets an allocation proportional to $\sigma_j n_j$. Many sampling methods for approximate query processing, such as the ones used in Chaudhuri et al. (2007); Agarwal et al. (2013), are based on Neyman allocation.

A problem with Neyman allocation is that it assumes that each stratum has abundant data, much larger than the size of samples. However, in practice, strata can be bounded, and may not always contain a large number of elements, and in such situations, Neyman allocation can be suboptimal. To see this, suppose there were 10 strata in the population, and suppose stratum 1

---

$^1$The standard deviation of data within a stratum is distinct from the variance of an estimate of an aggregate that is derived from a stratified random sample.
had 100 items and a standard deviation of 100, strata 2 to 10 each had 1000 items and a standard deviation of 0.1. With a sample size of $M = 1000$ items ($\approx 11\%$ of data size), Neyman allocation assigns 917 samples to stratum 1 and 9 samples each to the other strata. However stratum 1 has only 100 items, and it is wasteful to allocate more samples to this stratum. We call such strata, which have a small number of elements relative to the assigned sample size, as “bounded” strata. For instance, in our experiments with a sample size of 1 million from the one-year-long OpenAQ dataset OpenAQ (2019) on air quality measurements, we found that after the first month, 11 out of 60 strata are bounded. For data with bounded strata, Neyman allocation is clearly no longer the variance-optimal method for sample size allocation.

Another problem with the current state-of-the-art is that methods for SRS are predominantly offline methods, and assume that all data is available before sampling starts. As a result, systems that rely on SRS (e.g., Agarwal et al. (2013); Chaudhuri et al. (2007)) cannot easily adapt to new data arrivals and will need to recompute stratified random samples from scratch, as more data arrives. However, with the advent of streaming data warehouses such as Tidalrace Johnson and Shkapenyuk (2015), it is imperative to have methods for SRS that work on dynamic data streams and maintain stratified random samples in an incremental manner. In this work, we consider the general problem of variance-optimal SRS in both the offline and streaming settings, when some of the strata may be bounded.

### 4.2 Overview

#### 4.2.1 Preliminaries

We consider the construction and maintenance of a stratified random sample of data that is either stored offline or arriving as a stream. Stratified sampling can be viewed as being composed of three parts – stratification, sample allocation, and sampling.

Stratification is a partitioning of the universe into a number of disjoint strata, such that the union of all strata equals the universe. Equivalently, it is the assignment of each data element to a unique stratum. Stratification is often a pre-defined function of one or more attributes of the
data element. For example, the work of Chaudhuri et al. Chaudhuri et al. (2007) stratifies tuples within a database table based on the set of selection predicates in the query workload that the tuple satisfies. In the OpenAQ dataset OpenAQ (2019), air quality data measurements can be stratified on the basis of geographic location and measurement type, so that tuples relevant to a query can typically be composed of the union of strata. Our work assumes that the universe has already been partitioned into strata and that each tuple comes with a stratum identifier. This assumption fits the model assumed in Chaudhuri et al. (2007); Agarwal et al. (2013).

Our work deals with sample allocation, the task of partitioning the available memory budget of \( M \) samples among the strata. In the case of offline sampling, allocation needs to be done only once, after knowing the data in its entirety. In the case of streaming sampling, the allocation may need to be continuously re-adjusted as more data arrives, and the characteristics of different strata change.

The final sampling step chooses within each stratum, the assigned number of samples uniformly at random. In the case of offline stratified sampling, the sampling step can be performed in a second pass through the data after sample size allocation, using reservoir sampling on the subset of elements belonging to each stratum. In the case of streaming sampling, the sampling step is not as easy, since it needs to occur simultaneously with sample (re-)allocation, which may change the allocations to different strata over time.

**Variance-Optimal Allocation.** Given a data set, \( R = \{v_1, v_2, \ldots, v_n\} \) of size \( n \), whose elements are stratified into \( r \) strata, numbered 1, 2, \ldots, \( r \). For each \( i = 1 \ldots r \), let \( S_i \) be a uniform random sample of size \( s_i \) drawn without replacement from stratum \( i \). Let \( S = \{S_1, S_2, \ldots, S_n\} \) denote the stratified random sample.

The sample mean of each per-stratum sample \( S_i \) of size \( s_i \) is: \( \bar{y}_i = \frac{\sum_{v \in S_i} v}{s_i} \). The population mean of \( R \), \( \mu_R \) can be estimated as: \( \bar{y} = \frac{\sum_{i=1}^{r} n_i \bar{y}_i}{n} \), using the sample means of all strata. It can be shown that the expectation of \( \bar{y} \) equals \( \mu_R \).

Given a memory budget of \( M \leq n \) elements to store all the samples, so that \( \sum_i s_i = M \), we address the question: What is the value of each \( s_i \), the size of sample \( S_i \), so as to minimize the
variance of $\bar{y}$. The variance of $\bar{y}$ can be computed as follows (e.g. see Theorem 5.3 in Cochran (1977)):

$$V = V(\bar{y}) = \frac{1}{n^2} \sum_{i=1}^{r} n_i (n_i - s_i) \frac{\sigma_i^2}{s_i} = \frac{1}{n^2} \sum_{i=1}^{r} n_i^2 \sigma_i^2 \frac{1}{s_i} - \frac{1}{n^2} \sum_{i=1}^{r} n_i \sigma_i^2.$$  \hspace{1cm} (4.1)

We call the answer to this question as a variance-optimal allocation of sample sizes to different strata.

**Neyman Allocation for Strata that are abundant.** All previous studies on variance-optimal allocation assume that every stratum has a large volume of data, to fill its sample allocation. Under this assumption, Neyman allocation Neyman (1934); Cochran (1977) minimizes the variance $V$, and allocates a sample size for stratum $i$ as

$$M \cdot \frac{(n_i \sigma_i)}{\left( \sum_{j=1}^{r} n_j \sigma_j \right)}$$

Given a collection of data elements $R$, we say a stratum $i$ is abundant, if

$$n_i \geq M \cdot \frac{(n_i \sigma_i)}{\left( \sum_{j=1}^{r} n_j \sigma_j \right)}$$

Otherwise, the stratum $i$ is bounded. Clearly, Neyman allocation is optimal only if each stratum is abundant. It no longer be optimal if one or more strata are bounded. We consider the general case of variance-optimal allocation where there may be bounded strata.

**4.2.2 Solution Overview**

We note that both offline and streaming SRS can be viewed as a problem of “sample size reduction” in a variance-optimal manner. With offline SRS, we can initially view the entire data as a (trivial) sample of zero variance, where the sample size is very large – this sample needs to be reduced to fit within the memory budget of $M$ records. If this reduction is done in a manner that minimizes the increase of variance, the resulting sample is a variance-optimal sample of size $M$.

In the case of streaming SRS, the streaming algorithm maintains a current stratified random sample of size $M$. It also maintains the characteristics of each stratum, including the number of elements $n_i$ and standard deviation $\sigma_i$, in a streaming manner using $O(1)$ space per stratum.
When a set of new stream elements arrive, we can let the per-stratum reservoir sampling algorithms continue sampling as before. If the sample size increases due to this step, then we are again faced with a problem of sample size reduction – how can this be reduced to a sample of size $M$ in a variance-optimal manner?

Based on the above observation, we first present a variance-optimal sample size reduction method in Section 4.3. We start with an algorithm for reducing the size of the sample by one element, followed by a general algorithm for reducing the size by $\beta \geq 1$ elements, and then present an improved algorithm with a faster runtime. The variance-optimal offline algorithm VOILA can be viewed as an application of sample size reduction – details are presented in Section 4.4. We present a tight lower bound for any streaming algorithm, followed by S-VOILA, an algorithm for streaming SRS in Section 4.5. Note that the streaming algorithm S-VOILA does not necessarily lead to a variance-optimal sample. Though the individual sample-size reduction steps performed during observation of the stream are locally optimal, the overall result may not be optimal. Further details are in Section 4.5. We present a detailed experimental study of our algorithms in Section 4.6.

### 4.3 Variance-Optimal Sample Size Reduction

Suppose it is necessary to reduce an SRS of total size $M$ to an SRS of total size $M' < M$. This will need to reduce the size of the samples of one or more strata in the SRS. Since the sample sizes are reduced, the variance of the resulting estimate will increase. We consider the task of variance-optimal sample size reduction (VOR), i.e., how to partition the reduction in sample size among the different strata in such a way that the increase in the variance is minimized.

Consider Equation 4.1 for the variance of an estimate derived from the stratified random sample. Note that, for a given data set, a change in the sample sizes of different strata $s_i$ does not affect the parameters $n$, $n_i$, and $\sigma_i$. VOR can be formulated as the following non-linear program.

$$\text{Minimize} \quad \sum_{i=1}^{r} \frac{n_i^2 \sigma_i^2}{s_i'}$$

subject to constraints:

$$0 \leq s_i' \leq s_i \text{ for each } i = 1, 2, \ldots, r$$
\[
\sum_{i=1}^{r} s_i' = M'
\]

We observe that, without Constraint 4.3, and if all strata are unbounded, the answer to the above optimization program is exactly the Neyman allocation under memory budget \(M'\). However, we have to deal with the additional Constraint 4.3 and the possibility of a stratum being bounded, in an efficient manner. In the rest of this section, we present efficient approaches for computing the VOR.

### 4.3.1 Special Case: Reduction by One Element

We first present an efficient algorithm for the case where the size of a stratified random sample is reduced by one element. An example application of this case is in designing a streaming algorithm for SRS, when stream items arrive one at a time.

We introduce a terminology that we will use frequently in the rest of the paper. Given a memory budget \(M\), the Neyman allocation size for stratum \(i\) is \(M_i = M \cdot n_i \sigma_i / \sum_{j=1}^{r} n_j \sigma_j\). The task is to eliminate a random element from a stratum \(i\) such that after reducing the sample size \(s_i\) by one, the increase in variance \(V\) (Equation 4.1) is the smallest. Our solution is to choose stratum \(i\) such that the partial derivative of \(V\) with respect to \(s_i\) is the largest over all possible choices of \(i\).

\[
\frac{\partial V}{\partial s_i} = -\frac{n_i^2 \sigma_i^2}{n^2} \frac{1}{s_i^2}.
\]

We choose stratum \(\ell\) where:

\[
\ell = \arg \max_i \left\{ \frac{\partial V}{\partial s_i} \left| 1 \leq i \leq r \right\} \right\} = \arg \min_i \left\{ \frac{n_i \sigma_i}{s_i} \left| 1 \leq i \leq r \right\} \right\} = \arg \max_i \left\{ \frac{s_i}{M'_i} \left| 1 \leq i \leq r \right\} \right\}, \quad (4.3)
\]

where \(M'_i\) is the Neyman allocation size for stratum \(i\) under the new memory budget \(M'\). Equation 4.3 is due to the fact that each \(M'_i\) is proportional to \(n_i \sigma_i\). This gives the following lemma.

**Lemma 1.** When required to reduce the size of an stratified random sample by one, the increase in variance of the estimated population mean is minimized if we reduce the size of \(S_\ell\) by one, where

\[
\ell = \arg \min_i \left\{ \frac{n_i \sigma_i}{s_i} \left| 1 \leq i \leq r \right\} \right\}.
\]
Algorithm 1: SingleSSR(): Variance-Optimal Sample Size Reduction by One

\[
\text{return } \arg \min_i \left\{ \frac{n_i \sigma_i}{s_i} \mid 1 \leq i \leq r \right\} \quad \text{/* The id of the stratum whose sample size shall be reduced by one. */}
\]

In the case where we have multiple choices for \(\ell\) using Lemma 1, we choose the one where the current sample size \(s_\ell\) is the largest. Algorithm SingleSSR for reducing the sample size by one is shown in Algorithm 1. It is straightforward to observe that the run time of the algorithm is \(O(r)\).

4.3.2 Reduction by \(\beta \geq 1\) Elements

We now consider the general case, where the sample size needs to be reduced by some number \(\beta, 1 \leq \beta \leq M\). A possible solution idea is to repeatedly apply the one-element reduction algorithm (Algorithm 1 from Section 4.3.1) \(\beta\) times. Each iteration, a single element is chosen from a stratum such that the overall variance increases by the smallest amount. However, this greedy approach may not yield a sample with the smallest resulting variance. On the other hand, an exhaustive search of all possible evictions is not feasible either, since the number of possible ways to partition a reduction of size \(\beta\) among \(r\) strata is \(\binom{\beta + r - 1}{r}\), which is exponential in \(r\) and a high degree polynomial in \(\beta\), which can be very large. We now present efficient approaches to VOR. We first present a recursive algorithm, followed by a faster iterative algorithm. Before presenting the algorithm, we present the following useful characterization of a variance-optimal reduction.

**Definition 1.** We say that stratum \(i\) is oversized under memory budget \(M\), if its allocated sample size \(s_i > M_i\). Otherwise, we say that stratum \(i\) is not oversized.

**Lemma 2.** Suppose that \(E\) is the set of \(\beta\) elements that are to be evicted from a stratified random sample such that the variance \(V\) after eviction is the smallest possible. Then, each element in \(E\) must be from a stratum whose current sample size is oversized under the new memory budget \(M' = M - \beta\).

**Proof.** We use proof by contradiction. Suppose one of the evicted elements, is deleted from a sample \(S_\alpha\) such that the sample size \(s_\alpha\) is not oversized under the new memory budget. Because
the order of the eviction of the $\beta$ elements does not impact the final variance, suppose that element $e$ is evicted after the other $\beta - 1$ evictions have happened. Let $s_\alpha$ denote the size of sample $S_\alpha$ at the moment $t$ right after the first $\beta - 1$ evictions and before evicting $e$. The increase in variance caused by evicting an element from $S_\alpha$ is

$$\Delta = \frac{1}{n^2} \left( \frac{n_\alpha^2 \sigma_\alpha^2}{s_\alpha(s_\alpha - 1)} \right) = \left( \frac{\sum_{i=1}^r n_i \sigma_i}{nM'} \right)^2 \frac{M'^2_\alpha}{s_\alpha(s_\alpha - 1)} > \left( \frac{\sum_{i=1}^r n_i \sigma_i}{nM'} \right)^2$$

where $M'_\alpha = M' \frac{n_\alpha \sigma_\alpha}{\sum_{i=1}^r n_i \sigma_i}$ is the Neyman allocation for stratum $\alpha$ under memory budget $M'$. The last inequality is due to the fact that $S_\alpha$ is not oversized under budget $M'$ at time $t$, i.e., $s_\alpha \leq M'_\alpha$.

Note that an oversized sample exist at time $t$, since there are a total of $M' + 1$ elements in the stratified random sample at time $t$, and the memory target is $M'$. Instead of evicting $e$, if we choose to evict another element $e'$ from an oversized sample $S_{\alpha'}$, the resulting increase in variance will be:

$$\Delta' = \frac{1}{n^2} \left( \frac{n_{\alpha'}^2 \sigma_{\alpha'}^2}{s_{\alpha'}(s_{\alpha'} - 1)} \right) = \left( \frac{\sum_{i=1}^r n_i \sigma_i}{nM'} \right)^2 \frac{M'^2_{\alpha'}}{s_{\alpha'}(s_{\alpha'} - 1)} < \left( \frac{\sum_{i=1}^r n_i \sigma_i}{nM'} \right)^2$$

where $M'_{\alpha'} = M' \frac{n_{\alpha'} \sigma_{\alpha'}}{\sum_{i=1}^r n_i \sigma_i}$ is the Neyman allocation for stratum $\alpha'$ under memory budget $M'$. The last inequality is due to the fact that $S_{\alpha'}$ is oversized under budget $M'$ at time $t$, i.e., $s_{\alpha'} > M'_{\alpha'}$. Because $\Delta' < \Delta$, at time $t$, evicting $e'$ from $S_{\alpha'}$ leads to a lower variance than evicting $e$ from $S_\alpha$. This is a contradiction to the assumption that evicting $e$ leads to the smallest variance, and completes the proof. \[\square\]

Lemma 2 implies that it is only necessary to reduce the size of the samples that are oversized under the target memory budget $M'$. Samples that are not oversized can be given their current allocation, even under the new memory target $M'$. Our algorithm based on this observation first allocates sizes to the samples that are not oversized. The remaining memory now needs to be allocated among the oversized samples. Since this can again be viewed as a sample size reduction problem, while focusing on a smaller set of (oversized) samples, this is accomplished using a recursive call under a reduced memory budget; See Lemma 3 for a formal statement of this idea. The base case for this recursion is when all samples under consideration are oversized. In this case, we simply

Input: A – set of strata under consideration.
M – target sample size for all strata in A.

Output: For \( i \in A \), \( L[i] \) is the final size of sample for stratum \( i \).

1. \( O \leftarrow \emptyset \)  // oversized samples
2. for \( j \in A \) do
   3. \( M_j \leftarrow M \cdot n_j \sigma_j / \sum_{t \in A} n_t \sigma_t \)  // Neyman allocation if memory \( M \) divided among \( A \)
   4. if \( (s_j > M_j) \) then \( O \leftarrow O \cup \{j\} \)
   5. else \( L[j] \leftarrow s_j \)  // Keep current allocation
6. if \( O = A \) then
   7. // All samples oversized. Recursion stops.
      8. for \( j \in A \) do \( L[j] \leftarrow M_j \)
   9. else
      10. SSR(O, M - \( \sum_{j \in A - O} s_j \cdot L) \)

use the Neyman allocation to each stratum, under the reduced memory budget \( M' \) (Observation 1).

Our algorithm SSR is shown in Algorithm 2.

Let \( S = \{S_1, S_2, \ldots, S_r\} \) be the current stratified random sample. Let \( A \) denote the set of all strata under consideration, initialized to \( \{1, 2, \ldots, r\} \). Let \( O \) denote the set of oversized samples, under target memory budget for \( S \), and \( U = S - O \) denote the collection of samples that are not oversized. When the context is clear, we use \( O \), \( U \), and \( A \) to refer to the set of stratum identifiers as well as the set of samples corresponding to these identifiers.

**Lemma 3.** A variance-optimal eviction of \( \beta \) elements from \( S \) under memory budget \( M' \) requires a variance-optimal eviction of \( \beta \) elements from \( O \) under memory budget \( M' - \sum_{j \in U} s_j \).

**Proof.** Recall that \( s'_i \) denotes the final size of sample \( S_i \) after \( \beta \) elements are evicted. Referring to the variance \( V \) from Equation 4.1, we know a variance-optimal sample size reduction of \( \beta \) elements from \( S \) under memory budget \( M' \) requires to minimize

\[
\sum_{i \in A} \frac{n_i^2 \sigma_i^2}{s'_i} - \sum_{i \in A} \frac{n_i^2 \sigma_i^2}{s_i}
\] (4.4)
By Lemma 2, we know $s_i = s'_i$ for all $i \in \mathcal{U}$. Hence, minimizing Formula 4.4 is equivalent to minimizing

$$\sum_{i=O} n_i^2 \frac{\sigma_i^2}{s_i'} - \sum_{i \in \mathcal{O}} n_i^2 \frac{\sigma_i^2}{s_i}$$  \hspace{1cm} (4.5)

The minimization of Formula 4.5 is exactly the result obtained from a variance-optimal sample size reduction of $\beta$ elements from oversized samples under the new memory budget $M' - \sum_{i \in \mathcal{U}} s_i$. \hfill \Box

Observation 1. In the case every sample in the stratified random sample is oversized under target memory $M'$, i.e., $\mathcal{S} = \mathcal{O}$, the variance-optimal reduction is to reduce the size of each sample $S_i \in \mathcal{S}$ to its Neyman allocation $M'_i$ under the new memory budget $M'$.

The following theorem summarizes the correctness and time complexity of Algorithm SSR.

**Theorem 1.** Algorithm 2 (SSR) finds a variance-optimal reduction of the stratified random sample $\mathcal{A}$ under new memory budget $M$. The worst-case time of SSR is $O(r^2)$, where $r$ is the number of strata.

**Proof.** Correctness follows from Lemmas 2–3 and Observation 1. The worst-case time happens when each recursive call sees only one stratum that is not oversized. In such a case, the time of all recursions of SSR on a stratified random sample across $r$ strata is: $O(r + (r-1) + \ldots + 1) = O(r^2)$. \hfill \Box

Although SSR takes $O(r^2)$ time in the worst case, its time complexity tends to be much better in practice. If the number of samples that are not oversized contributes at least a certain percentage of the total number of samples being considered in every recursion, its overall time cost will be $O(r)$.

4.3.2.1 A faster implementation

We also present an iterative algorithm for sample size reduction, FastSSR, with time complexity $O(r \log r)$. FastSSR shares the same algorithmic foundation as SSR, but uses a faster method to find samples that are not oversized.
Algorithm 3: FastSSR($M$): A fast implementation of Sample Size Reduction without using recursion.

**Input:** The strata under consideration is implicitly \{1, 2, \ldots, $r$\}. $M$ is the target total sample size.

**Output:** For $1 \leq i \leq r$, $\mathcal{L}[i]$ is set to the final size of sample for stratum $i$, such that the increase of the variance $V$ is minimized.

1. Allocate $\mathcal{L}[1..r]$, an array of numbers
2. Allocate $Q[1..r]$, an array of ($x, y, z$) tuples
3. for $i = 1 \ldots r$ do $\quad Q[i] \leftarrow (i, n_i \sigma_i, s_i/(n_i \sigma_i))$;
4. Sort array $Q$ in ascending order on the $z$ dimension
5. for $i = (r - 1)$ down to 1 do
6. $\quad Q[i].y \leftarrow Q[i].y + Q[i + 1].y$
7. $\quad M_{\text{new}} \leftarrow M; D \leftarrow Q[1].y$
8. for $i = 1 \ldots r$ do
9. $\quad M_{Q[i],x} \leftarrow M \cdot n_{Q[i],x} \sigma_{Q[i],x}/D$
10. if $s_{Q[i],x} > M_{Q[i],x}$ then break
11. $\quad \mathcal{L}[Q[i],x] \leftarrow s_{Q[i],x}$
12. $\quad M_{\text{new}} \leftarrow M_{\text{new}} - s_{Q[i],x}$
   // Check the next sample, which must exist.
13. $\quad M_{Q[i+1],x} \leftarrow M \cdot n_{Q[i+1],x} \sigma_{Q[i+1],x}/D$
14. if $s_{Q[i+1],x} > M_{Q[i+1],x}$ then // oversized
15. $\quad M \leftarrow M_{\text{new}}; D \leftarrow Q[i + 1].y$
   // Reduce sample size to target.
16. for $j = i \ldots r$ do
17. $\quad \mathcal{L}[Q[j],x] \leftarrow M \cdot n_{Q[j],x} \sigma_{Q[j],x}/D$
18. return $\mathcal{L}$
Definition 2. Let $Q[1..r]$ be an array of $(x,y,z)$ tuples, where each $Q[i]$ is initialized as

$$(i, n_i \sigma_i, s_i/(n_i \sigma_i))$$

Array $Q$ is then sorted on its $z$ dimension.

Lemma 4. Under any given memory budget $M$, if there exists at least one unoversized sample, the collection of the identifiers of the unoversized samples must be occupying a continuous prefix of the array $Q$.

Proof. Recall that under a memory budget $M$, the Neyman allocation size for stratum $i$ is $M_i = n_i \sigma_i / D$, where $D = \sum_{i=1}^{r} n_j \sigma_j$. A sample $S_i$ is not oversized if and only if $s_i \leq M_i$, i.e., $s_i/(n_i \sigma_i) \leq 1/D$. A sample $S_i$ is oversized if and only if $s_i > M_i$, i.e., $s_i/(n_i \sigma_i) > 1/D$. Because array $Q$ is in the ascending order of its $z$ dimension, the lemma is proved.

Lemma 4 implies that we can linearly walk along the array $Q$ from $Q[1]$ toward $Q[r]$. By comparing the sample size and the Neyman allocation size for each stratum we are looking at during the walk, we will be able to find the collection of samples that are not oversized, under the new target memory budget $M'$.

After finding the prefix of the $Q$ array that represents the collection of samples that are not oversized, we pause the walk and then set the new memory $M'$ budget to be $M'$ minus the total size of the samples in the prefix. Then, we treat the remaining part (after excluding the prefix) of the array $Q$ as the current array $Q$ and do the same walk under the new memory budget $M'$.

The walk will stop if we do not see any sample that is not oversized under the current memory budget $M'$. In that case, we just set the size of the samples in the current array $Q$ to be their Neyman allocation size, under the current memory budget.

In order to avoid the recomputation of $D$, which is needed in computing the Neyman allocation, for every new memory budget during the walk, we precompute the $D$ for every suffix of the array $Q$ and save the result in the $y$ dimension of the $Q$ array.

The method FastSSR in Algorithm 3 shows the pseudocode of this faster algorithm for variance-optimal sample size reduction.
Theorem 2. (1) The FastSSR procedure in Algorithm 3 finds the correct size of each sample of an stratified random sample, whose memory budget is reduced to $M$, such that the increase of the variance $V$ is minimized. (2) The worst-case time cost of FastSSR on a stratified random sample across $r$ strata is $O(r \log r)$.

Proof. (1) The correctness of the procedure follows from Lemmas 2–3, Observation 1, and Lemma 4. (2) The time cost of FastSSR is dominated by the step of sorting array $Q$ on its $z$ dimension (Line 4), so the worst-case time cost of FastSSR is $O(r \log r)$. \qed

4.4 VOILA: Variance-Optimal Offline SRS

We now present an algorithm for computing the variance-optimal allocation of sample sizes in the case when one or more strata may be bounded. Note that the actual sampling step is straightforward for the offline algorithm – once the allocation of sample sizes is determined, the samples can be chosen in a second pass through the data, using reservoir sampling within each stratum. Hence, in the rest of this section, we focus on determining the allocation. Consider a static data set $R$ of $n$ elements across $r$ strata, where stratum $i$ has $n_i$ elements, and has standard deviation $\sigma_i$. How can a memory budget of $M$ elements be partitioned among the strata in a variance-optimal manner? We present VOILA (Variance-OptimaL Allocation), an efficient offline algorithm for variance-optimal allocation that can handle strata that are bounded. VOILA is a generalization of the classic Neyman allocation – in the case when every stratum has abundant data, it reduces to Neyman allocation.

The following two-step process reduces variance-optimal offline SRS to variance-optimal sample size reduction.

Step 1: Suppose we start with a memory budget of $n$. Then, we will just save the whole data set in the stratified random sample, and thus each sample size $s_i = n_i$. By doing so, the variance $V$ is minimized, since $V = 0$ (Equation 4.1). 
**Algorithm 4:** VOILA ($M$): Variance-optimal stratified random sampling for bounded data

**Input:** $M$ is the memory target

1. for $i = 1 \ldots r$ do
2. 
3. $s_i \leftarrow n_i$ // assume total available memory of $n$
4. $L \leftarrow$ FastSSR($M$)
5. return $L$ /* $L[i] \leq n_i$ is the sample size for stratum $i$ in a variance-optimal stratified random sample. */

**Step 2:** Given the stratified random sample from Step 1, we reduce the memory budget from $n$ to $M$ such that the resulting variance is the smallest. This can be done using variance-optimal sample size reduction, by calling SSR or FastSSR with target sample size $M$.

VOILA (Algorithm 4) simulates this process. The algorithm only records the sample sizes of the strata in array $L$, without creating the actual samples. The actual sample from stratum $i$ is created by choosing $L[i]$ random elements from stratum $i$, using any method for offline uniform random sampling without replacement.

**Theorem 3.** Given a data set $R$ with $r$ strata, and a memory budget $M$, VOILA (Algorithm 4) returns in $L$ the sample size of each stratum in a variance-optimal stratified random sample. The worst-case time cost of VOILA is $O(r \log r)$.

*Proof.* The correctness follows from the correctness of Theorem 2, since the final sample is the sample of the smallest variance that one could obtain by reducing the initial sample (with zero variance) down to a target memory of size $M$. The run time is dominated by the call to FastSSR, whose time complexity is $O(r \log r)$.

4.5 Streaming SRS

We now consider the maintenance of an SRS from a data stream, whose elements are arriving continuously.
4.5.1 A Lower Bound for Streaming SRS

Given a data stream $\mathcal{R}$ across $r$ strata, let $V^*$ denote the sample variance of the stratified random sample created by VOILA, using a memory budget of $M$. Because VOILA is variance optimal, $V^*$ is the smallest variance that we can get from any stratified random sample of $\mathcal{R}$ under the memory budget $M$. While VOILA is not a streaming algorithm, $V^*$ is a lower bound on the variance that a streaming algorithm can achieve, under memory budget $M$.

Let $V$ denote the sample variance of an SRS of $\mathcal{R}$ using the same memory budget $M$. We say $V$ is an approximation of $V^*$ with a multiplicative error of $\alpha$, for some constant $\alpha \geq 1$, if: (1) the sample within each stratum $i$ is chosen uniformly at random without replacement from stratum $i$. (2) $V \leq \alpha \cdot V^*$.

**Theorem 4.** Any streaming algorithm for maintaining an SRS over a stream with $r$ strata using a memory of $M$ records must, in the worst case, have a multiplicative error $\Omega(r)$ when compared with the optimal variance that can be achieved by a stratified random sample using memory of $M$ records.

**Proof.** We use proof by contradiction. Suppose that it is possible to maintain an approximate stratified random sample with a multiplicative error less than $r$.

Consider an input stream where the $i$th stratum consists of elements in the range $[i, i + 1)$, where the right endpoint of the stratum does not include $i + 1$. Suppose the stream so far has the following elements. For each $i$ from 1 to $r$, there are $(\alpha - 1)$ copies of element $i$ and one copy of $(i + \varepsilon)$ where $0 < \varepsilon < 1$ and $\alpha \geq 3$. After observing these elements, for each stratum $i$, $1 \leq i \leq r$, we have:

$$n_i = \alpha, \quad \mu_i = i + \frac{\varepsilon}{\alpha},$$

$$\sigma_i = \sqrt{\left(\frac{\alpha - 1}{\alpha}\right)^2 + \left(\frac{\varepsilon}{\alpha}\right)^2} \cdot \frac{1}{\alpha} = \frac{\sqrt{\alpha - 1}}{\alpha} \cdot \varepsilon.$$

Observe that, due to the memory budget $M$, at least one stratum has its sample size no more than $M/r$. Without loss of generality, let’s say that stratum is stratum 1.
Suppose an element of value \((2 - \varepsilon)\) arrives in the stream, where \(\varepsilon = 1/(r - 1)\). This element belongs to stratum 1. Let \(n'_1, \mu'_1, \sigma'_1\) denote the new size, mean, and standard deviation of stratum 1 after this element arrives.

\[
n'_1 = \alpha + 1, \quad \mu'_1 = 1 + \frac{1}{\alpha + 1},
\]

\[
\sigma'_1 = \sqrt{\frac{(\alpha - 1)\left(\frac{1}{\alpha + 1}\right)^2 + (\varepsilon - \frac{1}{\alpha + 1})^2 + (1 - \varepsilon - \frac{1}{\alpha + 1})^2}{\alpha + 1}} = \sqrt{\frac{\varepsilon^2 + (1 - \varepsilon)^2 - 1}{\alpha + 1}}.
\]

It follows that:

\[
(\alpha + 1)\sqrt{\frac{1 - \frac{1}{\alpha + 1}}{\alpha + 1}} \leq n'_1\sigma'_1 \leq (\alpha + 1)\sqrt{\frac{1 - \frac{1}{\alpha + 1}}{\alpha + 1}} \quad (4.6)
\]

\[
\Rightarrow \frac{\sqrt{\alpha}}{2} \leq n'_1\sigma'_1 \leq \sqrt{\alpha} \quad \text{(Note: } \alpha > 2\text{)} \quad (4.7)
\]

In 4.6, the left inequality stands when \(\varepsilon = 1/2\) and the right inequality stands when \(\varepsilon = 0\) or 1. We also have:

\[
\sum_{i=2}^{r} n_i\sigma_i = (r - 1)\alpha \frac{\sqrt{\alpha - 1}}{\alpha} - \varepsilon = \sqrt{\alpha - 1} \quad \left(\text{Note: } \varepsilon = \frac{1}{r - 1}\right)
\]

\[
\Rightarrow \frac{\sqrt{\alpha}}{2} \leq \sum_{i=2}^{r} n_i\sigma_i \leq \sqrt{\alpha} \quad \text{(Note: } \alpha > 2\text{)} \quad (4.8)
\]

Let \(V\) denote the sample variance of the stratified random sample maintained over the stream of \((r\alpha + 1)\) elements. Let \(V^*\) denote the smallest sample variance that one can get from a stratified random sample from these \((r\alpha + 1)\) date elements. Let \(\Delta = \left(n'_1\sigma'_1^2 + \sum_{i=2}^{r} n_i\sigma_i^2\right)/n^2\).

We observe the facts that (1) after processing these \((r\alpha + 1)\) elements, the sample size \(s_1 \leq M/r + 1\). (2) The portion of the sample variance contributed by strata 2, 3, \ldots, \(r\) is minimized if the memory budget for these strata, which is no more than \(M\), are equally shared, because all \(n_i\sigma_i\) are equal for \(i = 2, 3, \ldots, r\). Using these two facts and the definition of the sample variance in
40

On the other hand, the smallest sample variance \( V^\ast \) is achieved by using the Neyman allocation of the memory budget \( M \), assuming each stratum has sufficient data to fill its sample size assigned by the Neyman allocation. By Inequalities 4.7 and 4.8, we know that in the Neyman allocation for the current stream of \( r\alpha + 1 \) elements, stratum 1 uses at least \( M/3 \) memory space, whereas all other strata equally share at least \( M/3 \) memory space as well because all \( n_i\sigma_i \) are equal for \( i = 2, 3, \ldots, r \). Using these observations into Equation 4.1, we have:

\[
V^\ast \leq \frac{1}{n^2} \left( \frac{n_1^2\sigma_1^2}{M/3} + \sum_{i=2}^{r} \frac{n_i^2\sigma_i^2}{M/3(r-1)} \right) - \Delta
\]

\[
\leq \frac{1}{n^2} \left( \frac{\alpha}{M/3} + \sum_{i=2}^{r} \frac{(\alpha - 1)\varepsilon^2}{M/3(r-1)} \right) - \Delta
\]

\[
= \frac{1}{n^2} \frac{6\alpha - 3}{M} - \Delta \quad \text{(Note: } \varepsilon = \frac{1}{r - 1})
\]

Because \( \Delta \geq 0 \) and \( M > r \), we have:

\[
\frac{V}{V^\ast} \geq \frac{V + \Delta}{V^\ast + \Delta} \geq \Omega(r)
\]

The idea in the proof is to construct an input stream with \( r \) strata where the variance of all strata are the same until a certain point, where the variance of a single stratum increases to a high value – a variance-optimal SRS will respond by increasing the allocation to this stratum. However, a streaming algorithm is unable to do so quickly, since it is in general unable to collect enough samples to satisfy the increased allocation to this stratum. Though a streaming algorithm
is able to compute the variance-optimal allocation to different strata in an online manner, it cannot actually maintain these samples using limited memory.

We also note that the above lower bound is tight, since the simple uniform allocation, which allocates $M/r$ memory to each of the $r$ strata that have been observed so far, has a variance which is within a multiplicative factor of $r$ of the optimal. However, we see that the policy of uniform allocation performs poorly in practice, since it does not distinguish between different strata, whether based on volume or variance.

4.5.2 S-VOILA: Practical Streaming SRS

We now present S-VOILA, a practical streaming algorithm for stratified random sampling, that works for streams with zero or more bounded strata. Choices made by S-VOILA are “locally optimal” in the following sense: when new stream elements arrive, the decision of whether or not to select these elements (which will make it necessary to discard sampled elements from other strata) is made in a way that minimizes the variance of the estimate from resulting sample. S-VOILA can be viewed as an online version of VOILA, which constructs an SRS with minimal variance using a multi-pass algorithm through the entire data.

Let $R$ denote the stream so far, and $R_i$ the substream of elements belonging to stratum $i$. Within a single stratum, any algorithm for SRS needs to maintain a uniform random sample of all data seen so far. In streaming SRS, the memory $s_i$ allocated to a stratum $i$ may change with time, depending on the data arriving within this stratum, and other strata. One issue for a streaming algorithm is to maintain a uniform random sample within stratum $i$ when $s_i$ is changing. A decrease in the allocation $s_i$ can be handled easily, through discarding randomly chosen elements from the current sample $S_i$ until the desired sample size is reached. What if we need to increase the allocation to stratum $i$? If we simply start sampling new elements according to the higher allocation to stratum $i$, then recent elements in the stream will be favored over the older ones, and the sample within stratum $i$ is no longer uniformly chosen. In order to ensure that $S_i$ is always
Algorithm 5: S−VOILA: Initialization

**Input:** \( M \) – total sample size, \( r \) – number of strata.

// \( S_i \) is the sample for stratum \( i \), and \( R_i \) is the substream of elements from Stratum \( i \)

1. Load the first \( M \) stream elements in memory, and partition them into \( r \) per-stratum samples, \( S_1, S_2, \ldots, S_r \), such that \( S_i \) consists of \((e, d)\) tuples from stratum \( i \), where \( e \) is the element, \( d \) is the key of the element, chosen uniformly at random from \((0, 1)\).

2. For each stratum \( i \), compute \( n_i, \sigma_i \). Initialize \( d_i \leftarrow 1 \), where \( d_i \) is the smallest key among all elements in \( R_i \) not selected in \( S_i \).

Algorithm 5 presents the initialization of \( S−VOILA \), which simply loads the first \( M \) stream elements into the memory budget and divides them into \( r \) samples \( S_1, S_2, \ldots, S_r \), and initializes state. As new elements arrive, they change the frequency and the variance of a stratum and may lead to changes in the desired allocation of samples to strata. While it is possible to recompute the variance-optimal allocation, it is not possible to sample additional elements into strata as necessary, since we do not have the ability to look at all the data seen so far. However, our algorithm locally optimizes the variance through carefully selecting the strata from which samples will be discarded to make way for one or more incoming sampled elements.
Algorithm 6: S-VOILA: Process a new minibatch $B$ of $b$ elements. Note that $b$ need not be fixed, and can vary from one minibatch to the other.

1. $\beta \leftarrow 0$; // #selected elements in the minibatch
2. for each $e \in B$ do
3.   Let $\alpha$ denote the stratum of $e$
4.   Update $n_\alpha$ and $\sigma_\alpha$
5.   Assign a random key $d \in (0,1)$ to element $e$;
6.   if $d \leq d_\alpha$ then // element $e$ is selected
7.     $S_\alpha \leftarrow \{e\} \bigcup S_\alpha; \beta \leftarrow \beta + 1$
8.   /* Variance-optimal eviction of $\beta$ elements */
9.   if $\beta = 1$ then // faster for evicting 1 element
10.      $\ell \leftarrow \text{SingleSSR}();$
11.      Delete one element of largest key from $S_\ell$
12.      $d_\ell \leftarrow$ smallest key discarded from $S_\ell$
13. else if $\beta > 1$ then
14.      $L \leftarrow \text{FastSSR}(M)$;
15.      for $i = 1 \ldots r$ do // Actual element evictions
16.         if $L[i] < s_i$ then
17.             $d_i \leftarrow$ smallest key discarded from $S_i$

S-VOILA supports the insertion of a minibatch of any size $b \geq 1$, where the value of $b$ is even allowed to be dynamic during the execution of S-VOILA. When users fix $b = 1$, S-VOILA becomes streaming algorithm that handles one element at a time. As the value $b$ increases, we can expect S-VOILA to have a better variance, since its optimization decisions are based on greater amount of data. Algorithm 6 presents the algorithm for maintaining the stratified random sample when a new minibatch of multiple elements arrives. Lines 2–7 make one pass through the minibatch to update the statistics of each stratum and store the selected elements into the sample. If $\beta > 0$ elements from the minibatch get selected into the sample, in order to balance the memory budget at $M$, we will need to evict $\beta$ elements from the stratified random sample—this is accomplished using the variance-optimal sample size reduction technique from Section 4.3. For the special case where we only need to evict one element, we can use the faster algorithm SingleSSR (Lines 8–11); otherwise, FastSSR is used (Lines 12–17).
Lemma 5 shows that the sample maintained within by S-VOILA within each stratum is a uniform random sample, showing this is a valid stratified sample.

**Lemma 5.** For each \( i = 1, 2, \ldots, r \) sample \( S_i \) maintained by S-VOILA (Algorithm 6) is selected uniformly at random without replacement from stratum \( R_i \).

**Proof.** First, note that each \( S_i \) is selected from \( R_i \) without replacement, because each element of \( R_i \) is selected into \( S_i \) no more than once. Next, we prove the uniformity of \( S_i \). In case \( |S_i| = n_i \), all elements of \( R_i \) are in \( S_i \). In case \( |S_i| < n_i \), \( S_i \) contains the \( |S_i| \) elements with the smallest keys from stratum \( R_i \), because: (1) Anytime an element is discarded from \( S_i \), it is the element of the largest key in the sample. (2) If another element with key \( d \) enters later, it cannot be inserted into \( S_i \) unless \( d \) is smaller than or equal to all other keys discarded so far. Because the keys of elements are assigned randomly, each element has a chance of \( |S_i|/n_i \) to be selected into \( S_i \). Therefore, \( S_i \) is a uniform random sample from \( R_i \) without replacement. \( \square \)

**Theorem 5.** If the minibatch size \( b = 1 \), then the worst-case time cost of S-VOILA for processing an element is \( O(r) \). The expected time for processing an element belonging to stratum \( \alpha \) is \( O(1 + r \cdot s_\alpha/n_\alpha) \), which is \( O(1) \) when \( r \cdot s_\alpha = O(n_\alpha) \). If \( b > 1 \), then the worst-case time cost of S-VOILA for processing a minibatch is \( O(r \log r + b) \).

**Proof.** \( b = 1 \): The worst case happens when the single new element from belonging to stratum \( \alpha \) gets selected into \( S_\alpha \). In that case, we need to reduce the stratified random sample size by one via SingleSSR, which takes \( O(r) \) time. The probability that the new element is selected into \( S_\alpha \) is equal to \( s_\alpha/n_\alpha \), so the expected time follows.

\( b > 1 \): The time cost for Lines 2–7 is \( O(b) \). The time cost for Lines 8–17 is \( O(r \log r + \beta) \). So the total time cost is \( O(b) + O(r \log r + \beta) = O(r \log r + b) \). The per-element amortized time cost is \( O(1) \) when \( b = \Omega(r \log r) \) \( \square \)

We can expect S-VOILA to have an amortized per-item processing time of \( O(1) \) in many circumstances.
When \( b = 1 \): After observing enough stream elements from stratum \( \alpha \), such that \( r \cdot s_\alpha = O(n_\alpha) \), the expected processing time of an element becomes \( O(1) \). Even if certain strata have a very low frequency, the expected time cost for processing a single element is still expected to be \( O(1) \), because elements from an infrequent stratum \( \alpha \) are unlikely to appear in the minibatch.

When \( b > 1 \): The per-element amortized time cost of S-VOILA is \( O(1) \), when the minibatch size \( b = \Omega(r \log r) \).

### 4.6 Experimental Evaluation

The algorithms are evaluated on real-world data as well as synthetic data. The input is a stored set or a continuous stream of records from a data source, which is processed by the sampler which either outputs the sample at the end of computation (offline sampler) or continuously maintains a sample (streaming sampler). A streaming sampler must process data in a single pass, and is unable to access elements that were observed earlier, unless they are stored in memory. An offline sampler has access to all data received, and can compute a stratified random sample using multiple passes through data.

We evaluate the algorithms in two ways. The first is a direct evaluation of the quality of the samples, through the resulting allocation and the variance of an estimate of the population mean obtained using the samples. The second is through the accuracy of approximate query processing using the maintained samples.

#### 4.6.1 Sampling Methods

We implemented three offline sampling methods, each of which uses two passes to compute a stratified random sample. The first pass is to determine strata characteristics from which the sample size of each stratum is derived, and the second pass is to collect the samples. Each method is given the same total memory of \( M \) records. We implemented VOILA as described in the paper, and Neyman, that uses Neyman allocation. As explained earlier, Neyman will lead to a bounded stratum being allocated a greater sample size than the data within the stratum. This leaves some portion of
the total memory unused by Neyman. To improve upon this, we implemented an extended version of Neyman called Neyman+, which uses the entire memory allocation. Neyman+ first runs Neyman. Any unused memory is allocated equally among the remaining (non-bounded) strata. This may lead to more strata becoming bounded, and the process is continued recursively, until all the memory is used up.

We implemented the following stream sampling methods: S-VOILA with different minibatch sizes, reservoir sampling Reservoir, and Uniform – SRS with uniform allocation. Reservoir maintains a uniform random sample chosen without replacement from the stream - we expect the number of samples allocated to stratum $i$ by Reservoir to be proportional to $n_i$. Uniform allocates the same amount of memory to each stratum that has been observed. If a stratum has too few data points to fill its current allocation, then the remaining memory is allocated uniformly among other strata, and this memory redistribution may happen further, recursively.

For all experiments on comparing sampling methods based on allocations or on variance, each data point is the mean of five independent runs.

4.6.2 Data

We used two datasets. The first is the OpenAQ dataset OpenAQ (2019), which contains more than 31 million records of air quality measurements from 7,923 locations in 62 countries around the world.
Figure 4.2: Relative standard deviations of different strata, demonstrated by normalized cumulative standard deviations observed by the end of each month.

Figure 4.3: The number of strata seen so far, and the number of records in data, as a function of time.

world in 2016. The measurements includes particulate matter (PM10 and PM2.5), sulfur dioxide (SO$_2$), carbon monoxide (CO), nitrogen dioxide (NO$_2$), ozone (O$_3$), and black carbon (BC). Data is replayed in time order to generate the data stream and is stratified based on the country of origin and the type of measurement, e.g., all measurements of carbon monoxide in the USA belong to one stratum, all records of sulphur dioxide in India belong to another stratum, and so on. The total number of strata at the end of observation is 177, as shown in Figure 4.3.

We note that each stratum begins with zero records, and in the initial stages, each stratum is bounded. As more data are observed, many of the strata are not bounded anymore, but it is still the case that there are some strata with few observations, when compared with other strata. Further, new strata are added as more sensors are incorporated into the data stream. Figure 4.3 shows
Figure 4.4: Relative frequencies of different strata in the synthetic dataset change over time.

Figure 4.5: Relative standard deviations of different strata in the synthetic dataset change over time.

that new strata are being added with time. Figure 4.1 and 4.2 respectively show the cumulative frequency and standard deviation of the data over time. As seen, the relative frequency and relative standard deviation of different strata change significantly. As a result, the variance-optimal sample-size allocations to strata also change over time, and the streaming algorithms need to adapt to these changes.

The characteristics of real data, including number and properties of strata are changing frequently and continuously, and the allocation is a result of the combined adaptation due to multiple changes. In order to evaluate on data over which we have more control, we created a synthetic data source. Each record $i$ from this source is a tuple $\langle sid, val \rangle$ where $sid$ is the id of the stratum the record belongs to, $val$ is the value. The number of strata is set to 20. Frequencies are equal between strata, i.e., at any time, each stratum has approximately same amount of records. For a stratum $j$, the value of each record is drawn at random from Gaussian distribution with two parameters mean
\( \mu_j = 1 \) and standard deviation \( \sigma_j \). For the first 10,000 records, we set \( \sigma_j = 1 \) for all the strata. After that, we change the standard deviation of stratum 12 by setting \( \sigma_{12} = 20 \), while keeping the other strata fixed. Figures 4.4 and 4.5 show the relative frequencies and standard deviations of the synthetic dataset over time. While the frequencies are stable, the accumulated standard deviation shows how stratum 12 changes.

### 4.6.3 Allocations to Different Strata

We measured the allocation of samples to different strata. Unless otherwise specified, the sample size \( M \) is set to 1 million records. The allocation can be seen as a vector of numbers that sum up to \( M \) (or equivalently, normalized to sum up to 1), and we observe how this vector changes as more elements arrive. Figure 4.6 shows the allocations at a single point in time, at the end of September 2016, for OpenAQ data. From this figure, we see that the allocation of the streaming sampler S-VOILA tracks that of the variance-optimal offline sampler VOILA quite closely. As expected, Reservoir’s allocation is proportional to the volume of the stratum, while Uniform’s allocation is the same across all strata.

Figures 4.7, 4.8 and 4.9 show the change in allocations over time resulting from VOILA, S-VOILA with single element processing, and S-VOILA with minibatch processing (minibatch size = 1 day’s data). Visually, the allocations produced by the three methods track each other over time, showing that the streaming methods follow the allocation of the optimal offline algorithm, VOILA. To understand the difference between the allocations due to VOILA and S-VOILA quantitatively, we measure the cosine distance between the allocation vectors from VOILA and S-VOILA. The results show that allocation vectors due to S-VOILA and VOILA are very similar, since the cosine distance is close to
Figure 4.7: Allocation due to VOILA changes in allocation over time, OpenAQ data.

Figure 4.8: Allocation due to S-VOILA, with Single Element Processing, changes in allocation over time, OpenAQ data.

0 most of the time and less than 0.04 at all times. We further note that S-VOILA with minibatch processing yields an allocation that is closer to VOILA than S-VOILA with single element processing.

4.6.4 Comparison of Variance

We compared the variance of the estimates (Equation 4.1) from the stratified random samples produced by different methods, offline or streaming. The results are shown in Figures 4.11, 4.12 and 4.13. Generally, the variance of the sample due to each method increases over time, since the volume of data as well as the number of strata increase, while the sample size is fixed.

Among offline algorithms, we observe from Figure 4.11 that Neyman results in a variance that is larger than VOILA, by a factor of 1.4x to 50x. While Neyman is known to be variance-optimal
for unbounded strata, these results show that it is far from variance-optimal for bounded strata. VOILA is better than Neyman in two respects: (1) it uses all available memory, and (2) it allocates memory among strata in an optimal fashion. In order to measure the impact of the allocation, we compared the variance of VOILA with that of Neyman+, which uses all available memory. From Figure 4.11 we observe the following. First, VOILA always has a lower variance than Neyman+ and Neyman− note this also implies that at point in time, there are bounded strata in the OpenAQ data, since otherwise, Neyman would also result in optimal variance. Second, the variance due to VOILA is always smaller than the variance due to Neyman+, by a factor of 1.2x to 7.1x. This shows that carefully dealing with bounded strata using VOILA can lead to significantly better stratified random samples.
The comparison of the variance of streaming algorithms is shown in Figure 4.11. Among the streaming algorithms, we note that the variance due to S-VOILA with single element processing and with minibatch processing are typically close to that of the optimal algorithm, VOILA. The variance of S-VOILA using minibatch processing is very close to that of VOILA, showing that it is nearly variance-optimal at all times. The variance of S-VOILA with single element processing is typically worse than minibatch processing.

Figure 4.13 shows the relative difference between the variance produced by a streaming algorithm ($\hat{x}$) and the optimal variance due to VOILA ($x$), defined as $\frac{\hat{x} - x}{x}$. We note that the variance of both variants of S-VOILA are nearly equal to that of VOILA until March, when they start increasing relative to VOILA, and then converge back.
From analyzing the underlying data stream, we see that March is the time when a number of new strata appear in the data (Figure 4.3), causing significant changes in the optimal allocation of samples to strata (this can also be seen in Figure 4.10 showing the cosine distance between the allocations). An offline algorithm such as VOILA can resample more elements from a stratum, if necessary, since it has access to all data from the stratum. However, a streaming algorithm such as S-VOILA cannot do so and must wait for enough new elements to arrive in these strata before it can “catch up” to the allocation of VOILA. Hence, S-VOILA with single element as well as with minibatch processing start showing an increase in the variance at such a point. When data becomes stable again, and more elements arrive, the relative performance of S-VOILA improves. S-VOILA with minibatch processing approaches the optimal variance faster than S-VOILA with single element processing, which is as expected, since as the size of the minibatch increases, better optimization decisions are made with respect to which elements to exclude from the sample. In November and December, new strata appear again, and the relative performance is again affected. Overall, we note that S-VOILA with minibatch processing produces variance that is significantly closer to VOILA than S-VOILA with single element processing.

**Impact of Sample Size:** To understand the sensitivity to the size of the sample, we conducted an experiment where the sample size is varied from 5000 to 1 million records. We fixed the batch size to 100 thousand records. Figure 4.14 shows the snapshot in September 2016 of variances as a function of the sample size. Both VOILA and S-VOILA, with single element and minibatch...
processing, the variance decreases when the sample size increases. This is as expected, since larger samples produces better estimates of the population mean.

**Impact of Batch Size:** It is clear from Figure 4.12 that the variance of minibatch S-VOILA, where each batch contains data collected in a day, is significantly smaller than that of single element S-VOILA. In order to better understand the impact of the batch size, we conducted an experiment where we tried different batch sizes for minibatch streaming S-VOILA, chosen from \( \{1, 10, 10^2, 10^3, 10^4\} \). The results are shown in Figure 4.15. A batch size of 10 elements yields significantly better results than single element S-VOILA. A batch size of 100 or greater makes the variance of S-VOILA nearly equal to the optimal variance.
Figure 4.16: Query Performance as data size varies, with sample size fixed at 100,000. OpenAQ data.

4.6.5 Query Performance

We now evaluate the quality of these samples indirectly, through their use in approximate query processing, which is one of the major applications of sampling. The streaming sampler continuously maintains a stratified random sample of data (stored in memory), and use this sample to approximately answer aggregate queries, which are issued by the client. The offline sampler constructs its sample when needed, using VOILA, which takes two passes through the data. For evaluating the approximation error in query processing, we also implement an exact method for query processing, Exact, that stores every record in a table (stored in a MySQL database Widenius and Axmark (2002)) and answers a query using this table. While the exact method has zero error, its processing time is high, and so is its space overhead. Identical queries are made at the same time points in the stream to the different streaming and offline samplers, as well as to the exact query processor.

We measure the accuracy in query processing of the following samplers: Reservoir, Uniform, S-VOILA, VOILA, and Exact. We use the metric of relative error between the approximate answer and the exact answer, where the query asks for the mean of the data received across all strata. The sample size is set to 100,000 for all samplers. For S-VOILA, we set minibatch size to be 1, 100, and 10,000. Each data point is the mean of nine repetitions of the experiment with the same configuration.
Figure 4.17: Query Performance as sample size varies, with data size fixed at 21 million. OpenAQ data.

Figure 4.16 shows the relative error as the size of the streaming data increases, while the sample size is held fixed. The query was executed every three million element arrivals, up to thirty million, which covers the entire year of 2016 in the OpenAQ dataset. We note that the relative performance between different methods remains similar for most data sizes. Reservoir has a consistent errors since it is mainly affected by sample size rather than data size. Uniform is affected by total number of strata and as expected, we see an increasing error when the data size reaches 24 million, where the number of strata increases suddenly as shown in Figure 4.3, November 2016. The performance of VOILA and S-VOILA increase slightly with data size, though at much lower rates than Reservoir and Uniform. We note that S-VOILA with any minibatch size is very close to VOILA.

Figure 4.17 shows the impact of the sample size, as it varies from 5,000 to 100,000, and the queries were executed at a fixed time of stream to see how sample size would affect the accuracy of answering queries. As expected, all methods benefit from increased sample size. We observed S-VOILA and VOILA perform significantly better than Reservoir and Uniform even with smaller sample sizes. Another observation of S-VOILA is that a larger minibatch size does not always guarantee better accuracy. When total sample size is small, each stratum is allocated with a smaller space and there are fewer bounded strata. Therefore, the eviction made by single and minibatch processing affected the performance less. With our configuration, S-VOILA with minibatch ten thousand elements did not yield a better accuracy until sample size was set to one hundred thousand.
4.6.6 Adapting to a Change in Data Distribution

In a real-world dataset such as OpenAQ, the allocation is affected by the combination of multiple factors that continuously change. To better observe the behavior of our algorithms under a single change, we conducted an experiment with our synthetic data. Figure 4.5 shows a single change in stratum 12, where the standard deviation suddenly increases from 1 to 20 after the first 10,000 records are generated. Meanwhile, the standard deviation of all the other strata are stable and their frequencies are stable. After this change, we will expect Stratum 12 to be given a greater sample size than the other strata. The memory budget is set to 1,000 records, which is 2% of the data size at the end of the experiment.

Figures 4.18, 4.19, and 4.20 show the allocations produced by VOILA, single element S-VOILA, and minibatch S-VOILA, respectively. As seen, S-VOILA slowly captures the sudden change in the data by giving Stratum 12 more sample space over time. VOILA is more sensitive to the change,
due to the fact that VOILA works in an offline manner and is able to sample more data into Stratum 12 right after the change. Visually, minibatch S-VOILA is closer to the VOILA than single element S-VOILA.

Figure 4.21 shows the variance of different methods on synthetic data. At first, when the data is stable, all methods have nearly optimal variance. After a single change at 10,000 records, the variance of VOILA increases, while those of different versions of S-VOILA increase at a faster rate. S-VOILA with a higher minibatch size has a lower variance. Interestingly, the variance of all versions of S-VOILA converge to that of the optimal method, VOILA, though S-VOILA with a minibatch of 100 elements converges the fastest.

We also test the query performance of VOILA and S-VOILA with different minibatch sizes on synthetic data. Figure 4.22 shows the performance of a query across all strata. The first observation is VOILA is less affected by the distribution change since it samples from all the received data, while
S-VOILA methods had to discard data on the fly. Another observation is that performance of S-VOILA with a larger minibatch size will be closer to VOILA as stream continues.

4.7 Conclusions

We presented VOILA, a variance-optimal method for offline SRS from data that may have bounded strata. VOILA is a generalization of Neyman allocation, which assumes that each stratum has abundant data available. Our experiments show that on real and synthetic data, a stratified random sample obtained using VOILA can have a significantly smaller variance than one obtained by Neyman allocation. We also presented S-VOILA, an algorithm for streaming SRS with minibatch processing, whose sample allocation is continuously adjusted in a locally variance-optimal manner. Our experiments show that S-VOILA results in variance that is typically close to VOILA, which was given the entire input beforehand. The quality of the sample maintained by S-VOILA improves as the size of the minibatch increases. We show an inherent lower bound on the worst-case variance of any streaming algorithm for SRS – this limitation is not due to the inability to compute the optimal sample allocation in a streaming manner, but is instead due to the inability to increase sample sizes in a streaming manner, while maintaining uniformly weighted sampling within a stratum. There are several directions for future research, including (1) restratification in a streaming manner (2) incorporating time-decay into sampling, where more recent elements are given a higher
probability of being included in the sample, and (3) stratified random sampling on distributed data.
CHAPTER 5. OPTIMIZED SAMPLE SPACE ALLOCATION FOR GROUP-BY QUERIES

5.1 Introduction

As data size increases faster than the computational resources for query processing, answering queries on large data with a reasonable turnaround time has remained a significant challenge. One solution to handling this data deluge is through random sampling. A sample is a subset of data, collected with the use of randomness to determine which items are included in the sample and which are not. A query posed on the data can be quickly and approximately answered by executing the query on the sample, followed by an appropriate normalization. Sampling is attractive when a good trade-off can be obtained between the size of the sample and the accuracy of the answer.

We investigate random sampling to support “group-by” queries. A group-by query partitions an input relation into multiple groups according to the values of one or more attributes, and applies an aggregate function within each group. For instance, on a relation consisting of student records \(\text{Student}(\text{name}, \text{id}, \text{year}, \text{major}, \text{gpa})\), the following SQL query asks for the average \(\text{gpa}\) of each major:

\[
\text{SELECT major, } \text{AVG(gpa)} \\
\text{FROM Student} \\
\text{GROUP BY major}
\]

Group-by queries are very common, especially in data warehouses. For instance, in the TPC-DS benchmark \(\text{Nambiar and Poess (2006)}\), 81 out of 99 queries have group-by clause. We address the basic question: how to sample a table to accurately answer group-by queries?

A simple method is \textit{uniform sampling}, where each element is sampled with the same probability. It has been recognized that uniform sampling has significant shortcomings \(\text{Chaudhuri et al. (2001b)}\). Since a group will be represented in proportion to its volume (i.e. the number of elements in the
group), a group whose volume is small may have very few elements selected into the sample or may be missing altogether, while a group whose volume is large may have a large number of elements in the sample. This can clearly lead to high errors for some (perhaps a large fraction of) groups, as confirmed by our experimental study.

The accuracy of group-by queries can be improved using stratified sampling Acharya et al. (2000); Chaudhuri et al. (2007); Rösch and Lehner (2009). Data is partitioned into multiple strata. Uniform sampling is applied within each stratum, but different probabilities may be used across strata. The “best” way to use stratified sampling for group-by queries is not obvious – how to stratify the data into groups, and how to assign a sampling probability to a stratum?

Prior work on “congressional sampling” Acharya et al. (2000) has advocated the use of a fixed allocation of samples to each stratum, irrespective of its size (the “senate” strategy)\(^1\). Consider the above example, where the table is grouped by the attribute \texttt{major}. Suppose the population was stratified so that there is one stratum for each possible value of \texttt{major}, and uniform sampling is applied within each stratum. Even for this simple example, congressional sampling can lead to a sub-optimal allocation of samples. Consider two groups, 1 and 2 with the same number of elements and the same mean, but group 1 has a higher variance than group 2. In estimating the mean of each stratum, allocating equal numbers of samples to each group leads to a worse error for group 1 than for group 2. It is intuitively better to give more samples to group 1 than to group 2, but it is not clear how many more. \textit{Thus far, there has been no systematic study on sample allocation to groups, and our work aims to fill this gap.}

Our high-level approach is as follows. We first define a metric that quantifies the error of an estimate for a group-by query through using the sample, and we obtain an allocation and then a sample that minimizes this metric. Such an optimization approach to sampling has been used before. For instance, the popular “Neyman allocation” Neyman (1934); Nguyen et al. (2019) assigns samples to strata to minimize the variance of an estimate of the population mean that can be derived

\(^1\)More accurately, a hybrid of fixed and population-based allocations.
from the sample. The challenge with group-by queries is that there isn’t a single estimate to focus on, such as the population mean, but instead, an answer for each group.

Since multiple estimates are desired, we combine the metrics corresponding to different estimates to derive a global metric for the quality of a sample. One possible global metric is the sum of the variances of the different estimators. Such a global metric is inadequate in the case when the means of different groups are vastly different, in which case variances cannot be compared across groups. For instance, consider 2 groups with means $\mu_1 = 1000$ and $\mu_2 = 100$. Suppose an estimator for each group has a variance of 100. Though both estimators have the same variance, they are of different quality. If data within each group follows a normal distribution, then we can derive that with a probability greater than 96%, the estimated mean $y_1$ is between $[950; 1050]$ and the estimated mean $y_2$ is between $[50; 150]$. In other words, $y_1 = \mu_1 \pm 5\%$ and $y_2 = \mu_2 \pm 50\%$, and therefore $y_1$ is a much better estimate than $y_2$, in terms of relative accuracy. However, since the global metric adds up the variances, both the estimators contribute equally to the optimization metric. This leads to an allocation that favors groups that have a large mean over groups that have a small mean.

**Coefficient of Variation (CV):** In order to combine the accuracy of different estimators, perhaps with vastly different expectations, it is better to use the coefficient of variation (also called “relative variance”). The coefficient of variation of a random variable $X$ is defined as $CV[X] = \frac{S[X]}{E[X]}$, where $E[X]$ is the expectation (mean) and $S[X]$ the standard deviation of $X$, respectively. We assume that the attribute that is being aggregated has a non-zero mean, so the CV is well defined. The CV of a random variable $X$ is directly connected to its relative error $r(X) = |X - E[X]|/E[X]$ as follows. For a given $\epsilon > 0$, using Chebyshev’s inequality, we have $Pr[r(X) > \epsilon] = Pr[|X - E[X]| > \epsilon E[X]] \leq \frac{\text{Var}X}{\epsilon^2 E[X]^2} = \left(\frac{CV[X]}{\epsilon}\right)^2$. Smaller the CV, the smaller is the bound for the relative error of a given estimator. Our approach is to choose allocations to different strata so as to optimize a metric based on the CVs of different per-group estimators.
5.1.1 Contributions

(1) We present CVOPT, a novel framework for sampling from data which is applicable to any set of queries that result in multiple answers whose errors need to be simultaneously minimized (a specific example is a group-by query). CVOPT optimizes a cost function based on the weighted aggregate of the CVs of the estimates that are desired by the query. We present an algorithm (also called CVOPT) that computes a provably optimal allocation. To our knowledge, this is the first algorithm that results in a provably optimal sample for group-by queries. We consider two ways of aggregating CVs – one based on the $\ell_2$ norm of the CVs, and another based on the $\ell_\infty$ norm (the maximum) of the CVs.

(2) CVOPT can be adapted to the following circumstances, in increasing order of generality:

- With a single aggregation on a single method of grouping attributes (SASG), CVOPT leads to good statistical properties over a range of workloads. The distribution of errors of different groups is concentrated around the mean error, much more so than prior works Rösch and Lehner (2009); Acharya et al. (2000). As a result, the expected errors of the different per-group estimates are approximately equal, while prior work can lead to some groups being approximated very well while other groups being approximated poorly.

- With multiple aggregations for the same group-by clause (MASG), and for the general case of multiple aggregations and multiple ways of group-by (MAMG), we provide a way to derive a sample that optimizes for a combined metric on all aggregate queries. A special case is the Cube query that is widely used in analytics and decision support workloads.

- The user is allowed to specify a weight to each answer, allowing her to prioritize different query/group combinations, and use (perhaps uncertain) knowledge about the workload that may be specified by a probability distribution.
(3) The samples can be reused to answer various queries that incorporate selection predicates that are provided at query time, as well as new combinations of groupings.

(4) We present a detailed experimental study on two real-world datasets (OpenAQ and Bikes) that show that CVOPT using the $\ell_2$ norm of the CVs provides good quality estimates across all groups in a group-by query, and provides a relative error that is often up to $5\times$ smaller than prior work Acharya et al. (2000); Ding et al. (2016); Rösch and Lehner (2009). Further, no prior work can compete with CVOPT as a consistently second best solution for group-by queries.

Roadmap: We present preliminaries in Section 5.2, followed by algorithms for sampling for a single group-by query in Section 5.3 and algorithms for multiple group-by in Section 5.4. We present an algorithm for different error metrics in Section 5.5, a detailed experimental study in Section 5.6, followed by a survey of other related works, and the conclusions.

5.2 Preliminaries

For a random variable $X$, let $E[X]$ denote its expectation, $\text{VAR}[X]$ its variance, $S[X] = \sqrt{\text{VAR}[X]}$ its standard deviation, and $\text{CV}[X] = \frac{S[X]}{E[X]}$ its coefficient of variation.

The answer to a group-by query can be viewed as a vector of results, one for the aggregate on each group. In case multiple aggregates are desired for each group, the answer can be viewed as a two-dimensional matrix, with one dimension for the groups, and one for the aggregates. In this work, for simplicity of exposition, we focus on the aggregate $\text{AVG}$, i.e. the mean. Note that the sample can answer queries involving selection predicates provided at runtime (by simply applying the predicate on the sample) so that it is not possible to precompute the results of all possible queries. Aggregates such as median and variance can be handled using a similar optimization, and the same sample can be jointly optimized for multiple aggregate functions.

Let the groups of a given query be denoted $1, 2, \ldots, r$, and $\mu_1, \mu_2, \ldots, \mu_r$ denote the mean of value within each group. A sample of a table is a subset of rows of the table. From a sample, we

\footnote{Arbitrary new groupings and selection predicates are not supported with a provable guarantee. Indeed, we can show no sample that is substantially smaller than the data size can provide accurate answers for arbitrary queries, since one can reconstruct the original table by making such repeated queries.}
derive estimators $y_i$, of $\mu_i$, for each $i = 1 \ldots r$. We say that $y_i$ is an unbiased estimator of $\mu_i$ if $E[y_i] = \mu_i$. Note that an unbiased estimator does not necessarily mean an estimator that reliably estimates $\mu_i$ with a small error.

For a group-by query with $r$ groups in data, numbered 1..$r$, the aggregates can be viewed as an array $\mu = [\mu_1, \mu_2, \ldots, \mu_r]$, the estimates can be viewed as the vector $y = [y_1, y_2, \ldots, y_r]$, and coefficients of variation of the estimates as the vector $C = [CV[y_1], CV[y_2], \ldots, CV[y_r]]$. We first focus on the overall error equal to the $\ell_2$ norm of the vector $C$, defined as:

$$error(y) = \ell_2(C) = \sqrt{\sum_{i=1}^r (CV[y_i])^2}$$

Applying the above metric to the earlier example, we have $CV[y_1] = 10/1000 = 0.01$, while $CV[y_2] = 10/100 = 0.1$. This (correctly) evaluates $y_2$ as having a higher contribution to the overall error than $y_1$. If we were to optimize $\ell_2$ norm of the vector $[CV[y_1], CV[y_2]]$, resources would be spent on making $CV[y_2]$ smaller, at the cost of increasing $CV[y_1]$. We argue this is the right thing to do, since all results in a group-by query are in some sense, equally important. If we know apriori that some results are more important than others, they can be handled by using a weighting function for results, as we describe further. We also consider the $\ell_\infty$ norm, defined as:

$$\ell_\infty(C) = \max_{i=1}^r CV[y_i].$$

We can also assign weights to different results in computing the error. Consider a set of positive real valued numbers, one for each result $i = 1 \ldots r$, $w = \{w_i\}$. The weighted $\ell_2$ metric is:

$$error(y, w) = \ell_2(C, w) = \sqrt{\sum_{i=1}^r w_i \cdot (CV[y_i])^2},$$

where a larger $w_i$ indicates a higher accuracy demand for $y_i$.

### 5.3 Single Group-by

#### 5.3.1 Single Aggregate, Single Group-By

The first case is when we have a single aggregate query, along with a single group-by clause. Note that grouping does not have to use a single attribute, but could use multiple attributes. For example:
SELECT year, major, AVG(gpa)
FROM Student
GROUP BY year, major

Given a budget of sampling $M$ records from a table for a group-by query with $r$ groups, how can one draw a random sample such that the accuracy is maximized?

We use stratified sampling. In the case of a single group-by clause, stratification directly corresponds to the grouping. There is a stratum for each group, identified by a distinct value of the combination of group-by attributes. In the above example, there is one stratum for each possible combination of the (year, major) tuple. Probabilities of selecting a record from the table are not necessarily equal across different strata, but are equal within a stratum.

One simple solution, which we call as SENATE (used as a component in CS Acharya et al. (2000)), is to split the budget of $M$ records equally among all strata, so that each stratum receives $M/r$ samples. While this is easy to implement and improves upon uniform sampling, this solution has the following drawback. Consider two groups 1 and 2 with the same means $\mu_1 = \mu_2$, but with very different standard deviations within the groups, i.e. $\sigma_1 \gg \sigma_2$. Intuitively, it is useful to give more samples to group 1 than to group 2, to reduce the variance of the estimate within group 1. However, SENATE gives the same number of samples to both, due to which the expected quality of the estimate for group 1 will be much worse than the estimate for group 2. Intuitively, we need to give more samples to group 1 than group 2, but exactly how much more – this is answered using our optimization framework.

Before proceeding further, we present a solution to an optimization problem that is repeatedly used in our work.

**Lemma 6.** Consider positive valued variables $s_1, s_2, \ldots, s_k$ and positive constants $M, \alpha_1, \alpha_2, \ldots, \alpha_k$.

The solution to the optimization problem: minimize $\sum_{i=1}^{k} \frac{\alpha_i}{s_i}$ subject to 

$$\sum_{i=1}^{k} s_i \leq M$$

is given by $s_i = M \cdot \frac{\sqrt{\alpha_i}}{\sum_{j=1}^{k} \sqrt{\alpha_j}}$. 


Proof. Let \( s = [s_1, s_2, \ldots, s_k] \). Let \( f(s) = \sum_{i=1}^{k} \frac{\alpha_i}{s_i} \), and \( g(s) = (\sum_{i=1}^{k} s_i) - M \). We want to minimize \( f(s) \) subject to the constraint \( g(s) = 0 \). Using Lagrange multipliers:

\[
L(s_1, s_2, \ldots, s_k, \lambda) = f(s_1, s_2, \ldots, s_k) + \lambda g(s_1, s_2, \ldots, s_k)
\]

For each \( i = 1 \ldots r \), we set \( \frac{\partial L}{\partial s_i} = 0 \). Thus \( -\frac{\alpha_i}{s_i^2} + \lambda = 0 \), leading to \( s_i = \frac{\sqrt{\alpha_i}}{\sqrt{\lambda}} \). By setting \( g(s) = 0 \), we solve for \( \lambda \) and get \( s_i = M \cdot \frac{\sqrt{\alpha_i}}{\sum_{j=1}^{k} \sqrt{\alpha_j}} \).

We now consider how to find the best assignment of sample sizes to the different strata, using an optimization framework. Let \( s = [s_1, s_2, \ldots, s_r] \) denote the vector of assignments of sample sizes to different strata.

Theorem 6. For a single aggregation and single group-by, given weight vector \( w \) and sample size \( M \), the optimal assignment of sample sizes is to assign to group \( i \in \{1, 2, \ldots, r\} \) a sample size

\[
s_i = M \frac{\sqrt{w_i \sigma_i / \mu_i}}{\sum_{j=1}^{r} \sqrt{w_j \sigma_j / \mu_j}}
\]

Proof. Consider the estimators \( y = [y_1, y_2, \ldots, y_r] \) computed using the sample. Our objective is the \( \ell_2 \) error, which requires us to minimize \( \sqrt{\sum_{i=1}^{r} w_i \cdot (\text{CV}[y_i])^2} \), which is equivalent to minimizing \( \sum_{i=1}^{r} w_i \cdot (\text{CV}[y_i])^2 \).

The standard deviation of \( y_i \) depends on \( n_i \), the size of the \( i \)th group, \( s_i \), the size of the sample assigned to the \( i \)th group, and \( \sigma_i \), the standard deviation of the values in the \( i \)th group. By standard results on sampling, e.g. Theorem 2.2 in Cochran (1977), we have

\[
\text{CV}[y_i] = \frac{1}{\mu_i} \sqrt{\frac{\sigma_i^2 (n_i - s_i)}{n_i s_i}}
\]

Thus, we reduce the problem to minimizing \( \sum_{i=1}^{r} \frac{w_i \sigma_i^2 / \mu_i^2}{s_i} \cdot \sigma_i^2 (n_i - s_i) / n_i s_i \). Since \( r, \sigma_i, \mu_i, \) and \( n_i \) are fixed, this is equivalent to minimizing \( \sum_{i=1}^{r} \frac{w_i \sigma_i^2 / \mu_i^2}{s_i} \) subject to the condition \( \sum_{i=1}^{r} s_i = M \). Using Lemma 6, and setting \( \alpha_i = w_i \sigma_i^2 / \mu_i^2 \), we see that \( s_i \) should be proportional to \( \sqrt{w_i \sigma_i / \mu_i} \). \( \square \)
Algorithm 7: CVOPT-SASG: Algorithm computing a random sample for a single aggregate, single group-by.

Input: Database Table $T$, group-by attributes $A$, aggregation attribute $d$, weight vector $w$, memory budget $M$.

Output: Stratified Random Sample $S$

1. Let $A$ denote all possibilities of assignments to $A$ that actually occur in $T$. i.e. all strata.
   Let $r$ denote the size of $A$, and suppose the strata are numbered from 1 till $r$.
2. For each $i = 1 \ldots r$, compute the mean and variance of all elements in stratum $i$ along attribute $d$, denoted as $\mu_i, \sigma_i$ respectively. Let $\gamma_i \leftarrow \sqrt{w_i \sigma_i / \mu_i}$
3. $\gamma \leftarrow \sum_{i=1}^{r} \gamma_i$
4. for $i = 1 \ldots r$ do
   5. $s_i \leftarrow M \cdot \gamma_i / \gamma$
   6. Let $S_i$ be formed by choosing $s_i$ elements from stratum $i$ uniformly without replacement, using reservoir sampling
7. return $S = [S_1, S_2, \ldots, S_r]$

The above leads to algorithm CVOPT-SASG for drawing a random sample from a table $T$, described in Algorithm 7.

5.3.2 Multiple Aggregates, Single Group-by

We next consider the case of multiple aggregations using the same group-by clause. Without loss of generality, suppose the columns that were aggregated are columns $1, 2, \ldots, t$. As before, suppose the groups are numbered $1, 2, \ldots, r$. For group $i, 1 \leq i \leq r$ and aggregation column $j, 1 \leq j \leq t$, let $\mu_{i,j}, \sigma_{i,j}$ respectively denote the mean and standard deviation of the values in column $j$ within group $i$. Let $n_i$ denote the size of group $i$, and $s_i$ the number of samples drawn from group $i$. Let $y_{i,j}$ denote the estimate of $\mu_{i,j}$ obtained through the sample, and $\text{CV} [y_{i,j}] = \frac{S[y_{i,j}]}{\mu_{i,j}}$ denote the coefficient of variation of $y_{i,j}$. Further suppose that we are given weights for each combination of group and aggregate, which reflect how important these are to the user. Let $w_{i,j}$ denote the weight of the combination group $i$ and aggregation column $j$. Our minimization metric is a weighted combination of the coefficients of variation of all the $r \cdot t$ estimates, one for each group and aggregate combination.

\footnote{In the absence of user-input weights, we can assume default weights of 1.}
error(y, w) = √ \sum_{j=1}^{t} \sum_{i=1}^{r} w_{i,j} \cdot (\text{CV}[y_{i,j}])^2

**Theorem 7.** Given weights \( w \), and total sample size \( M \), the optimal assignment of sample sizes \( s \) among \( r \) groups is to assign to group \( i = 1, 2, \ldots, r \) sample size \( s_i = M \frac{\sqrt{\alpha_i}}{\sum_{i=1}^{r} \sqrt{\alpha_i}} \), where \( \alpha_i = \sum_{j=1}^{t} \frac{w_{i,j} \sigma_{i,j}^2}{\mu_{i,j}^2} \).

**Proof.** We use an approach similar to Theorem 6. Let \( y = \{y_{i,j}\}_{1 \leq i \leq r, 1 \leq j \leq t} \) denote the matrix of estimators for the means of the multiple aggregates, for different groups. Using the metric of \( \ell_2 \) error, we have to minimize \( \ell_2(\text{CV}[y], w) = \sqrt{\sum_{i=1}^{r} \sum_{j=1}^{t} w_{i,j} \cdot (\text{CV}[y_{i,j}])^2} \), which is equivalent to minimizing

\[
\sum_{i=1}^{r} \sum_{j=1}^{t} w_{i,j} \cdot \left( \frac{S[y_{i,j}]}{\mu_{i,j}} \right)^2.
\]

We note that \( S[y_{i,j}] = \sqrt{\frac{\sigma_{i,j}^2(n_i - s_i)}{n_is_i}} \), where \( \sigma_{i,j} \) is the standard deviation of the \( j \)th column taken over all elements in group \( i \). Thus, we reduce the problem to minimizing

\[
\sum_{i=1}^{r} \sum_{j=1}^{t} w_{i,j} \cdot \frac{\sigma_{i,j}^2(n_i - s_i)}{n_is_i} \cdot \frac{1}{\mu_{i,j}^2}.
\]

Since \( r, t, \sigma_{i,j}, \mu_{i,j}, \) and \( n_i \) are fixed, this is equivalent to minimizing

\[
\sum_{i=1}^{r} \sum_{j=1}^{t} w_{i,j} \frac{\sigma_{i,j}^2}{\mu_{i,j}^2 s_i} = \sum_{i=1}^{r} \frac{1}{s_i} \sum_{j=1}^{t} w_{i,j} \frac{\sigma_{i,j}^2}{\mu_{i,j}^2} = \sum_{i=1}^{r} \frac{\alpha_i}{s_i} \tag{5.1}
\]

subject to the condition \( \sum_{i=1}^{r} s_i \leq M \). Using Lemma 6, we arrive that the optimal assignment is

\[ s_i = M \frac{\sqrt{\alpha_i}}{\sum_{i=1}^{r} \sqrt{\alpha_i}}. \]

In our formulation, a weight can be assigned to each result in the output, reflecting how important this number is. For instance, if there are \( r \) groups and \( t \) aggregates desired, then there are \( r \times t \) results in the output, and the user can assign a weight for each result, \( w_{ij} \) for \( i = 1 \ldots r \) and \( j = 1 \ldots t \). A useful special case is when all weights are equal, so that all results are equally important to the user. If the user desires greater accuracy for group 1 when compared to group 2, this can be done by setting weights \( w_{1,*} \) to be higher than the weights \( w_{2,*} \), say 10 versus 1. The value of weight can also be deduced from a query workload, as we discuss in Section 5.4.3.
5.4 Multiple Group-Bys

Suppose that we had multiple attribute combinations on which there are group-by clauses. For instance, we may have a query where the student data is being grouped by major, and one query where it is being grouped by year, and another query where data is grouped by major as well as year. The additional challenge now is that there are multiple ways to stratify the data, to further apply stratified sampling. For instance, we can draw a stratified sample where data are stratified according to major only, or one where data are stratified according to year, or one where data are stratified according to both major and year. Any of these three samples can be used to answer all three queries, but may lead to high errors. For instance, a stratified sample where data is stratified according to year of graduation may lead to poor estimates for a group-by query based on major, since it may yield very few tuples or may completely miss some majors.

Our solution is to pursue a “finest stratification” approach where the population is stratified according to the union of all group-by attributes. In the above example, this leads to stratification according to a combination of major and year, leading to one stratum for each distinct value of the pair (year, major). This will serve group-by queries based solely on major or year, or a combination of both. The number of samples assigned to each stratum in such a stratification is determined in a principled manner.

5.4.1 Single Aggregate, Multiple Group-By

We first consider the case of a single aggregate and multiple group-bys, starting with the case of two group-bys and then extend to more than two group-bys. Suppose two queries $Q_1$ and $Q_2$ that aggregate on the same column, using different sets of group-by attributes, $A$ and $B$, respectively. Note that $A$ and $B$ need not be disjoint. For example $A$ can be (major, year) and $B$ can be (major, zipcode). If we combined the sets of group-by attributes, we get attribute set $C = A \cup B$. In the above example, $C$ is (major, year, zipcode). Let $A, B, C$ denote the set of all values possible for attributes in $A$, $B$, and $C$ respectively. Note that only those combinations that actually occur within data are considered.
Our algorithm based on finest stratification stratifies data according to attribute set \( C \), leading to a stratum for each combination of the values of attributes \( c \in C \). Samples are chosen uniformly within a single stratum, but the sampling probabilities in different strata may be different. Our goal is not to get a high-quality estimate for aggregates within each stratum according to \( C \). Instead, it is to get a high-quality estimate for aggregates for each group in \( A \) (query \( Q_1 \)) and in \( B \) (query \( Q_2 \)). We translate the above goal into an objective function that will help assign sample sizes to each stratum in \( C \).

For each stratum \( c \in C \), let \( s_c \) denote the number of samples assigned to this stratum, \( S_c \) the sample, \( \mu_c \) the mean of the aggregation column, and \( \sigma_c \) the standard deviation of the aggregation column. Let the sample mean for this stratum be denoted as \( y_c = \frac{\sum_{v \in S_c} v}{s_c} \). As \( C = A \cup B, A \in C \). For an assignment \( a \in A \) and an assignment \( c \in C \), we say \( c \in a \) if the attributes in set \( A \) have the same values in \( a \) and \( c \). Let \( C(a) \) denote the set of all \( c \in C \) such that \( c \in a \). For any \( c \in C \), let \( \Pi(c, A) \) denote the unique \( a \in A \) such that \( c \in C(a) \). Similarly, define \( \Pi(c, B) \).

For query \( Q_1 \), for group \( a \in A \), let \( \mu_a \) denote the mean of aggregate column, and \( n_a \) denote the size of the group. We desire to estimate \( \mu_a \) for each \( a \in A \). Suppose the estimate for \( \mu_a \) is \( y_a \). Similarly, we define \( \mu_b \), \( n_b \), and \( y_b \) for each group \( b \in B \). Our objective function is the weighted \( \ell_2 \) norm of \( \{ \text{CV} [y_a] | a \in A \} \cup \{ \text{CV} [y_b] | b \in B \} \), i.e.

\[
\sqrt{\sum_{a \in A} w_a \cdot (\text{CV} [y_a])^2 + \sum_{b \in B} w_b \cdot (\text{CV} [y_b])^2}
\]

The estimates for each group are derived as \( y_a = \frac{\sum_{c \in C(a)} n_c y_c}{\sum_{c \in C(a)} n_c} \), and similarly \( y_b = \frac{\sum_{c \in C(b)} n_c y_c}{\sum_{c \in C(b)} n_c} \). Using standard results from stratified sampling Cochran (1977), we have: \( \mathbb{E} [y_a] = \mu_a \), and

\[
\text{VAR} [y_a] = \frac{1}{n_a^2} \sum_{c \in C(a)} \left[ \frac{n_c^2 \sigma_c^2}{s_c} - n_c \sigma_c^2 \right]
\]

**Lemma 7.** The optimal assignment of sample sizes that minimizes the weighted \( \ell_2 \) norm of the coefficients of variation of the estimates is: for \( d \in C \) the sample size is \( s_d = M \cdot \frac{\sqrt{\beta_d}}{\sum_{c \in C} \sqrt{\beta_c}} \), where:

\[
\beta_c = n_c^2 \sigma_c^2 \left[ \frac{w_{\Pi(c,A)}}{n_{\Pi(c,A)}^2 \mu_{\Pi(c,A)}^2} + \frac{w_{\Pi(c,B)}}{n_{\Pi(c,B)}^2 \mu_{\Pi(c,B)}^2} \right]
\]
Proof. Our objective function is the weighted \( \ell^2 \) norm of the coefficients of variance of all estimators \( \{y_a | a \in A\} \) and \( \{y_b | b \in B\} \), which we want to minimize over all possibilities of the sample sizes \( s = \{s_c | c \in C\} \), subject to \( \sum_{c \in C} s_c = M \). Equivalently, we minimize the square of the weighted \( \ell^2 \) norm:

\[
Y(s) = \sum_{a \in A} w_a \cdot (\text{CV}[y_a])^2 + \sum_{b \in B} w_b \cdot (\text{VAR}[y_b])^2
\]

Using the exp. and variance of \( y_a \), we can rewrite \( Y(s) \):

\[
Y(s) = \sum_{a \in A} w_a \cdot \sum_{c \in C(a)} \left[ \frac{n_c^2 \sigma_c^2}{s_c} - n_c \sigma_c^2 \right] \frac{n_a^2 \mu_a^2}{n_a^2 \mu_a^2} + \sum_{b \in B} w_b \cdot \sum_{c \in C(b)} \left[ \frac{n_c^2 \sigma_c^2}{s_c} - n_c \sigma_c^2 \right] \frac{n_b^2 \mu_b^2}{n_b^2 \mu_b^2}
\]

This is equivalent to minimizing:

\[
Y'(s) = \sum_{a \in A} \sum_{c \in C(a)} w_a \frac{n_c^2 \sigma_c^2}{s_c n_c^2 \mu_a^2} + \sum_{b \in B} \sum_{c \in C(b)} w_b \frac{n_c^2 \sigma_c^2}{s_c n_b^2 \mu_b^2}
\]

\[
= \sum_{c \in C} \left[ \frac{w_{\Pi(c,A)} n_c^2 \sigma_c^2}{s_c n_c^2 \mu_{\Pi(c,A)}^2} + \frac{w_{\Pi(c,B)} n_c^2 \sigma_c^2}{s_c n_c^2 \mu_{\Pi(c,B)}^2} \right]
\]

We note that the problem turns to: minimize \( Y'(s) = \sum_{c \in C} \beta_c \) subject to \( \sum_{c \in C} s_c = M \). Using Lemma 6, we arrive that the optimal assignment of sample size is \( s_c = M \cdot \frac{\sqrt{\beta_c}}{\sum_{d \in C} \sqrt{\beta_d}} \).

An example: Consider a query \( Q_1 \) that groups by major and aggregates by gpa, and another query \( Q_2 \) that groups by year, and aggregates by gpa. Suppose each group in each query has the same weight 1. The above algorithm stratifies according to the \((\text{major}, \text{year})\) combination. For a stratum where major equals \( m \) and year equals \( y \), sample size \( s_{m,y} \) is proportional to:

\[
\beta_{m,y} = n_{m,y}^2 \sigma_{m,y}^2 \left[ \frac{1}{n_{m,y}^2 \mu_{m,y}^2} + \frac{1}{n_{*,y}^2 \mu_{*,y}^2} \right]
\]

Where \( n_{m,y}, n_{m,*}, n_{*,y} \) are respectively the number of elements with major = \( m \) and year = \( y \), the number of elements with major = \( m \), and the number of elements with year = \( y \), respectively.

Similarly for \( \mu_{m,y}, \mu_{m,*}, \mu_{*,y} \).
Example 2: Consider a query $R_1$ that groups by major, year and aggregates by gpa, and another query $R_2$ that groups by zipcode, year, and aggregates by gpa. Suppose all groups in both queries share the same weight 1. The above algorithm asks to stratify according to (major, zipcode, year) combination. For a stratum where major equals $m$, zipcode equals $z$ and year equals $y$, sample size $s_{m,z,y}$ is proportional to:

$$
\beta_{m,z,y} = n_{m,z,y}^2 \sigma_{m,z,y}^2 \left[ \frac{1}{n_{m,y}^2 \mu_{m,y}^2} + \frac{1}{n_{z,y}^2 \mu_{y,z,y}^2} \right]
$$

Where $n_{m,z,y}$, $n_{m,y}$, and $n_{z,y}$ are respectively the number of elements with major equal to $m$ and zipcode equal to $z$ and year equal to $y$, the number of elements with major equal to $m$ and year equal to $y$, and the number of elements with zipcode equal to $z$ and year equal to $y$, respectively. Similarly for $\mu_{m,z,y}$, $\mu_{m,y}$, $\mu_{z,y}$.

Generalizing to Multiple Group-Bys Suppose there were multiple group-by queries with attribute sets $A_1, A_2, \ldots, A_k$. The algorithm stratifies according to attribute set $C = \bigcup_{i=1}^{k} A_i$. For $i = 1 \ldots k$, let $A_i$ denote the universe of all assignments to attributes in $A_i$ and $C$ the universe of all possible assignments to attributes in $C$. Note that only those assignments that exist in the data need be considered. Extending the above analysis for the case of two group-bys, we get that the optimal assignment of samples as follows. For each $c \in C$, stratum $c$ is assigned sample size proportional to the square root of:

$$
\beta_c = n_c^2 \sigma_c^2 \sum_{i=1}^{k} \frac{w_{\Pi(c,A_i)}}{n_{\Pi(c,A_i)}^2 \mu_{\Pi(c,A_i)}^2}
$$

The proof of above result is similar to the case of Lemma 7. We minimize the $\ell_2$ norm:

$$
Y(s) = \sum_{a_i \in A_i} w_{a_i} \sum_{c \in C(a_i)} \frac{n_c^2 \sigma_c^2}{s_c n_{a_i}^2 \mu_{a_i}^2} \left[ \frac{n_c^2 \sigma_c^2}{s_c n_{a_i}^2 \mu_{a_i}^2} - \frac{n_c^2 \sigma_c^2}{n_{a_i}^2 \mu_{a_i}^2} \right]
$$

This is equivalent to minimizing:

$$
Y'(s) = \sum_{a_i \in A_i} \sum_{c \in C(a_i)} \frac{w_{a_i} n_c^2 \sigma_c^2}{s_c n_{a_i}^2 \mu_{a_i}^2}
$$

$$
= \sum_{c \in C} \sum_{i=1}^{k} \frac{w_{\Pi(c,A_i)} n_c^2 \sigma_c^2}{s_c n_{\Pi(c,A_i)}^2 \mu_{\Pi(c,A_i)}^2} = \sum_{c \in C} \beta_c
$$

Using Lemma 6, we have the result proved.
Cube-By Queries  An important special case of multiple group-by queries, often used in data warehousing, is the cube-by query. The cube-by query takes as input a set of attributes $A$ and computes group-by aggregations based on the entire set $A$ as well as every subset of $A$. For instance, if $A$ was the set major, year and the aggregation column is $A$, then the cube-by query poses four queries, one grouped by major and year, one grouped by only major, one grouped by only year, and the other without a group-by (i.e. a full table query). Our algorithm for multiple group-by can easily handle the case of a cube-by query and produce an allocation that optimizes the $\ell_2$ norm of the CVs of all estimates. We present an experimental study of cube-by queries in Section 5.6.

5.4.2 Multiple Aggregates, Multiple Group-Bys

Suppose two queries, $Q_1, Q_2$, that aggregate on the different columns $d_1$ and $d_2$ and also use different sets of group-by attributes, $A$ and $B$ that may be overlapping. e.g., $Q_1$ can aggregate gpa grouped by (major, year) and $Q_2$ can aggregate credits grouped by (major, zipcode).

We stratify the data according to attribute set $C = A \cup B$. As in Section 5.4.1, let $A, B, C$ denote the set of all values possible for attributes in $A$, $B$, and $C$ respectively. Also, for $a \in A$, $b \in B$, $c \in C$, let $C(a)$, $C(b)$, $\Pi(c, A)$ and $\Pi(c, B)$ be defined as in Section 5.4.1.

For each $c \in C$, let $n_c$ denote the number of data elements in this stratum, $\sigma_{c,1}$ the variance of the $d_1$ column among all elements in this stratum, and $\sigma_{c,2}$ the variance of the $d_2$ column in this stratum. Let $s_c$ denote the number of samples assigned to this stratum, and $y_{c,1}$ and $y_{c,2}$ respectively denote the sample means of the columns $d_1$ and $d_2$ among all elements in stratum $c$ respectively.

For each $a \in A$, we seek to estimate $\mu_{a,1}$, the mean of the $d_1$ column among all elements in this group. The estimate, which we denote by $y_{a,1}$ is computed as $\frac{\sum_{c \in C(a)} n_c y_{c,1}}{\sum_{c \in C(a)} n_c}$. Similarly for each $b \in B$, we seek to estimate $\mu_{b,2}$ the mean of the $d_2$ column among all elements in this group. Let $y_{b,2}$ be this estimate. Our optimization metric is the weighted $\ell_2$ norm of the coefficients of
variation of all estimates:

\[ \mathcal{L} = \sum_{a \in A} w_{a,1} \cdot (\text{CV}[y_{a,1}])^2 + \sum_{b \in B} w_{b,2} \cdot (\text{CV}[y_{b,2}])^2 \]

**Lemma 8.** For two group-by and two aggregates, the optimal assignment of sample sizes that minimizes the weighted $\ell_2$ norm of the coefficients of variation of the estimates is: for $d \in \mathcal{C}$ the sample size is $s_d = M \cdot \frac{\sqrt{\beta_d}}{\sum_{c \in \mathcal{C}} \sqrt{\beta_c}}$, where

\[ \beta_c = n_c^2 \left[ \frac{w_{\Pi(c,A)} n_c^2 \sigma_{c,1}^2}{n_{\Pi(c,A)}^2 \mu_{\Pi(c,A),1}^2} + \frac{w_{\Pi(c,B)} n_c^2 \sigma_{c,2}^2}{n_{\Pi(c,B)}^2 \mu_{\Pi(c,B),2}^2} \right] \]

**Proof.** Our objective function is the weighted $\ell_2$ norm of the coefficients of variance of all estimators \{\(y_{a,1}|a \in \mathcal{A}\) and \{\(y_{b,2}|b \in \mathcal{B}\), which we want to minimize over all possibilities of the vector of sample sizes $s = \{s_c|c \in \mathcal{C}\}$, subject to $\sum s_c = M$. Equivalently, we minimize the square of the weighted $\ell_2$ norm:

\[ Y(s) = \sum_{a \in \mathcal{A}} w_{a,1} \cdot (\text{CV}[y_{a,1}])^2 + \sum_{b \in \mathcal{B}} w_{b,2} \cdot (\text{CV}[y_{b,2}])^2 \]

Using the expected value and variance of $y_a$, we can rewrite $Y(s)$:

\[ Y(s) = \sum_{a \in \mathcal{A}} \sum_{c \in \mathcal{C}(a)} \left[ \frac{n_c^2 \sigma_{c,1}^2}{s_c} - n_c \sigma_{c,1}^2 \right] \mu_{a,1}^2 + \sum_{b \in \mathcal{B}} \sum_{c \in \mathcal{C}(b)} \left[ \frac{n_c^2 \sigma_{c,2}^2}{s_c} - n_c \sigma_{c,2}^2 \right] \mu_{b,2}^2 \]

This is equivalent to minimizing:

\[ Y'(s) = \sum_{a \in \mathcal{A}} \sum_{c \in \mathcal{C}(a)} \frac{w_{a,1} n_c^2 \sigma_{c,1}^2}{s_c n_{a,1}^2 \mu_{a,1}^2} + \sum_{b \in \mathcal{B}} \sum_{c \in \mathcal{C}(b)} \frac{w_{b,2} n_c^2 \sigma_{c,2}^2}{s_c n_{b,2}^2 \mu_{b,2}^2} \]

\[ = \sum_{c \in \mathcal{C}} \left[ \frac{w_{\Pi(c,A)} n_c^2 \sigma_{c,1}^2}{s_c n_{\Pi(c,A)}^2 \mu_{\Pi(c,A),1}^2} + \frac{w_{\Pi(c,B)} n_c^2 \sigma_{c,2}^2}{s_c n_{\Pi(c,B)}^2 \mu_{\Pi(c,B),2}^2} \right] \]

\[ = \sum_{c \in \mathcal{C}} \frac{\beta_c}{s_c} \]

Subject to $\sum s_c = M$. Using Lemma 6, we arrive that the optimal assignment of sample size is $s_c = M \cdot \frac{\sqrt{\beta_c}}{\sum_{d \in \mathcal{C}} \sqrt{\beta_d}}$. \qed
Table 5.1: An example Student table

<table>
<thead>
<tr>
<th>id</th>
<th>age</th>
<th>GPA</th>
<th>SAT</th>
<th>major</th>
<th>college</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>3.4</td>
<td>1250</td>
<td>CS</td>
<td>Science</td>
</tr>
<tr>
<td>2</td>
<td>22</td>
<td>3.1</td>
<td>1280</td>
<td>CS</td>
<td>Science</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
<td>3.8</td>
<td>1230</td>
<td>Math</td>
<td>Science</td>
</tr>
<tr>
<td>4</td>
<td>28</td>
<td>3.6</td>
<td>1270</td>
<td>Math</td>
<td>Science</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>3.5</td>
<td>1210</td>
<td>EE</td>
<td>Engineering</td>
</tr>
<tr>
<td>6</td>
<td>23</td>
<td>3.2</td>
<td>1260</td>
<td>EE</td>
<td>Engineering</td>
</tr>
<tr>
<td>7</td>
<td>27</td>
<td>3.7</td>
<td>1220</td>
<td>ME</td>
<td>Engineering</td>
</tr>
<tr>
<td>8</td>
<td>26</td>
<td>3.3</td>
<td>1230</td>
<td>ME</td>
<td>Engineering</td>
</tr>
</tbody>
</table>

We can generalize this to the case of more than two aggregations, and/or more than two group-bys. Suppose there were $k$ group-by queries $Q_1, Q_2, \ldots, Q_k$, with attribute sets $A_1, A_2, \ldots, A_k$. Each query $Q_i$ has multiple aggregates on a set of columns denoted as $\mathcal{L}_i$. In this case, the algorithm stratifies according to attribute set $C = \bigcup_{i=1}^{k} A_i$. For $i = 1 \ldots k$, let $\mathcal{A}_i$ denote the universe of all assignments to attributes in $A_i$ and $\mathcal{C}$ the universe of all possible assignments to attributes in $C$. Note that only those assignments that exist in the data need be considered. Extending the analysis from Section 5.3.2, 5.4.1, and 5.4.2, we get that the optimal assignment of samples is as follows. For each $c \in \mathcal{C}$, stratum $c$ is assigned sample size proportional to the square root of

$$\beta_c = \left( \frac{1}{n_c^2} \sum_{i=1}^{k} \left( \frac{1}{n_{H(c,A_i),\ell}} \sum_{\ell \in \mathcal{L}_i} \frac{w_{H(c,A_i),\ell} \cdot \sigma_{c,\ell}^2}{\mu_{H(c,A_i),\ell}^2} \right) \right).$$

5.4.3 Using A Query Workload

How can one use (partial) knowledge of a query workload to improve sampling? A query workload is a probability distribution of expected queries, and can be either collected from historical logs or created by users based on their experience. In the presence of a query workload, we show how to construct a sample that is optimized for this workload. We focus on the case of multiple aggregations, multiple group-by (MAMG), as others such as SASG and MASG are special cases. Our approach is to use the query workload to deduce the weights that we use in the weighted optimization for group-by queries.
Table 5.2: An example query workload on the Student table

<table>
<thead>
<tr>
<th>Queries</th>
<th>Repeats</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>SELECT AVG(age), AVG(gpa)   FROM Student GROUP BY major</td>
</tr>
<tr>
<td>B</td>
<td>SELECT AVG(age), AVG(sat)   FROM Student GROUP BY college</td>
</tr>
<tr>
<td>C</td>
<td>SELECT AVG(gpa) FROM Student GROUP BY major WHERE college=Science</td>
</tr>
</tbody>
</table>

Table 5.3: Aggregation groups and their frequencies produced from the example workload (Table 5.2)

<table>
<thead>
<tr>
<th>Aggregation groups</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>(age, major=CS), (age, major=Math), (age, major=EE)</td>
<td>25</td>
</tr>
<tr>
<td>(age, major=ME), (GPA, major=EE ), (GPA, major=ME)</td>
<td>35</td>
</tr>
<tr>
<td>(GPA, major=CS), (GPA, major=Math)</td>
<td>10</td>
</tr>
<tr>
<td>(age, college=Science), (age, college=Engineering)</td>
<td></td>
</tr>
<tr>
<td>(SAT, college=Science), (SAT, college=Engineering)</td>
<td></td>
</tr>
</tbody>
</table>

**An Example:** Consider an example Student table and its query workload shown in Tables 5.1 and 5.2. The workload has 45 group-by queries, of which three are distinct, named A, B, and C. Each group-by query stratifies its aggregation columns into a collection of mutually disjoint aggregation groups. Each aggregation group is identified by a tuple of \((a, b)\), where \(a\) is the aggregation column name and \(b\) is one value assignment to the group-by attributes. For example, Query A stratifies the age column into four aggregation groups: \((age, major=CS)\), \((age, major=Math)\), \((age, major=EE)\), and \((age, major=ME)\). Each aggregation group is a subset of elements in the aggregation column, e.g., aggregation group \((age, major=CS)\) is the set \(\{25, 22\}\). One aggregation group may appear more than once, because one query may occur multiple times and different queries may share aggregation group(s). Our preprocessing is to deduce all the aggregation groups and their frequencies from the workload. Table 5.3 shows the result of the example workload. We then use each aggregation group’s frequency as its weight in the optimization framework for CVOPT sampling.
5.5 CVOPT-INF and Extensions

We now consider minimizing for the maximum of the CVs, or the $\ell_\infty$ norm of all $CV[y_i]$ for different groups $i$. That is:

$$\ell_\infty(C) = \max_{i=1}^{\pi} CV[y_i] = \max_{i=1}^{\pi} \frac{\sigma_i}{\mu_i} \sqrt{\frac{n_i - s_i}{n_is_i}}$$

subject to the constraint $\sum_{i=1}^{\pi} s_i \leq M$. One obvious benefit of using $\ell_\infty(C)$ as the objective function is that the relative errors of different groups are expected to be the same.

The above problem has integrality constraints and is hard to optimize exactly. In the rest of this section, we present an efficient algorithm that relaxes the integrality constraints and assumes that $s_i$s can have real values. Note that we assume every $\sigma_i > 0$, any group $i$ where $\sigma_i = 0$ can be treated as a special case, since all its values are equal, and there is no need to maintain a sample of that group.

An efficient algorithm: Consider a collection of $r$ functions

$$f_i(x) = \frac{\sigma_i}{\mu_i} \sqrt{\frac{n_i - x}{n_is_i}}, \quad i = 1, 2, \ldots, r$$

where $x \in [0, M]$ is a real number. Our goal is find an assignment of $x = x_i$ for each $f_i$, that minimizes

$$\max_{i=1}^{r} f_i(x_i) = \max_{i=1}^{r} \frac{\sigma_i}{\mu_i} \sqrt{\frac{n_i - x_i}{n_is_i}}$$

and $\sum_{i=1}^{r} x_i \leq M$. Let $x_1^*, x_2^*, \ldots, x_r^*$ denote such assignments that solve this continuous optimization problem.

**Lemma 9.** $f_1(x_1^*) = f_2(x_2^*) = \ldots = f_r(x_r^*)$

*Proof.* We use proof by contradiction. Suppose the claim in the lemma is not true, then without losing generality, say $f_1(x_1^*), f_k(x_k^*), \ldots, f_k(x_k^*)$, for some $k < r$, are all the largest and $f_r(x_r^*)$ is the smallest. Then, because each $f_i$ is a strictly decreasing function, we can always reduce the value of each $x_i^*$, $i \leq k$, for some amount $c$ and increase the value of $x_r^*$ by amount $kc$, such that (1) every $f_i$
is decreased, \( i \leq k \). (2) \( f_r \) is increased, and (3) \( \max_{i=1}^{r} f_i \) is decreased. This is a contradiction, since we already know \( \max_{i=1}^{r} f_i(x_i^*) \) was minimized.

Following Lemma 9, we can have

\[
\frac{\sigma_1}{\mu_1} \sqrt{\frac{n_1 - x_1^*}{n_1 x_1^*}} = \frac{\sigma_2}{\mu_2} \sqrt{\frac{n_2 - x_2^*}{n_2 x_2^*}} = \ldots = \frac{\sigma_r}{\mu_r} \sqrt{\frac{n_r - x_r^*}{n_r x_r^*}}
\]

\[
\implies \frac{d_1}{x_1^*/\bar{x}_1^*} = \frac{d_2}{x_2^*/\bar{x}_2^*} = \ldots = \frac{d_r}{x_r^*/\bar{x}_r^*} \tag{5.2}
\]

where \( d_i = \frac{(\sigma_i/\mu_i)^2}{n_i} \) and \( \bar{x}_i^* = n_i - x_i^* \). Let \( D = \sum_{i=1}^{r} d_i \). By Equations 5.2, each \( x_i^*/\bar{x}_i^* \) is proportional to \( d_i \), i.e, \( x_i^*/\bar{x}_i^* = q^* \cdot d_i/D \) for some real number constant \( q^* \in [0, 1] \). Namely,

\[
x_i^* = \frac{q^* \cdot d_i/D}{1 + q^* \cdot d_i/D} n_i.
\]

Our approach to minimize \( \ell_{\infty}(C) \) is to perform a binary search for the largest integer \( q \in [0, n] \) that approximates \( q^* \), in such

\[
\sum_{i=1}^{r} x_i = \sum_{i=1}^{r} \frac{q \cdot d_i/D}{1 + q \cdot d_i/D} n_i \leq M.
\]

If the binary search returns \( q = 0 \), we set \( q = 1 \). We then assign each \( s_i = \left\lceil \frac{x_i}{\sum_{j=1}^{r} x_j} M \right\rceil \). Clearly, the total time cost for finding the set of \( s_i \) values is \( O(r \log n) \): There are a total of \( O(\log n) \) binary search steps where each step takes \( O(r) \) time.

**Extension to Other Aggregates.** Thus far, our discussion of the CVOPT framework has focused on group-by queries using the AVG aggregates (COUNT and SUM are very similar). CVOPT can be extended to other aggregates as well. To use the framework for an aggregate, we need to:

1. have the per-group CV of the aggregate of interest well defined, and
2. ensure that it is possible to compute the CV of a stratum using statistics stored for strata in finer stratification of this stratum. Hence, the method can potentially be extended to aggregates such as per-group median and variance.
5.6 Experimental Evaluation

We evaluate CVOPT for approximate query processing using Hive Thusoo et al. (2009) as the underlying data warehouse. The first phase is an offline sample computation that performs two passes of the data. The first pass computes some statistics for each group, and the second pass uses these statistics as input for CVOPT to decide the sample sizes for different groups and to draw the actual sample.

In the second phase, the sample obtained using CVOPT is used to (approximately) answer queries. The sample from CVOPT is representative and can answer queries that involve selection predicates provided at query time, as well as new combinations of groupings, without a need for recomputing the sample at query time. Overall, we can expect the overhead of the offline sampling phase to be small when compared with the time saved through sample-based query processing.

We collected two real-world datasets, OpenAQ and Bikes. OpenAQ OpenAQ (2019) is a collection of the air quality measurements of different substances, such as carbon monoxide, sulphur dioxide, etc. The data consists of about 200 million records, collected daily from ten thousand locations in 67 countries from 2015 to 2018. Bikes is data logs from Divvy Trips (2018), a Chicago bike share company. Customers can pick up a bike from one Divvy station kiosk and return it to any station at their convenience. The dataset contains information about these bike rides and also has some user information, such as gender or birthday. We analyzed all subscribers’ data from 2016 to 2018, for a total of approximately 11.5 million records. The datasets are stored in the database as 2 tables OpenAQ and Bike. Throughout this section, we introduce queries to those 2 tables, annotated with “AQ” and “B” prefixes, respectively.

Approximate answers from sample table are compared to the ground truth, derived using an exact computation from the full data. Let \( x \) and \( \bar{x} \) be the ground-truth and approximate answer, respectively. We use the relative error \( |\bar{x} - x|/x \) as the error metric for each group.

We implemented algorithms Uniform, Sample+Seek, CS and RL to compare with CVOPT. Uniform is the unbiased sampler, which samples records uniformly without replacement from the base table. RL is the algorithm due to Rosch and Lehner Rösch and Lehner (2009). CS is congressional sampling
Query AQ 1  Changing of bc overtime for each country.

WITH bc18 AS (  
  SELECT country, AVG(value) AS avg_value,  
  COUNT_IF(value > 0.04) AS high_cnt  
  FROM openaq  
  WHERE parameter = 'bc'  
  AND YEAR(local_time) = 2018  
  GROUP BY country ),  
bc17 AS (  
  SELECT country, AVG(value) AS avg_value,  
  COUNT_IF(value > 0.04) AS high_cnt  
  FROM openaq  
  WHERE parameter = 'bc'  
  AND YEAR(local_time) = 2017  
  GROUP BY country )  
SELECT country,  
  bc18.avg_value - bc17.avg_value AS avg Incre,  
  bc18.high_cnt - bc17.high_cnt AS cnt Incre  
FROM bc18 JOIN bc17  
ON bc18.country = bc17.country

Query AQ 2  MASP query to OpenAQ table.

SELECT country, parameter, unit,  
  SUM(value) agg1, COUNT(*) agg2  
FROM OpenAQ  
GROUP BY country, parameter, unit

algorithm due to Acharya et al. (2000). Sample+Seek is from Ding et al. (2016), after applying appropriate normalization to get an unbiased answer. CVOPT is the implementation of our $\ell_2$ optimal sampler. We also report results from CVOPT-INF, the $\ell_\infty$-optimal sampler. Unless otherwise specified, each method draws a 1% sample. Each reported result is the average of 5 identical and independent repetitions of an experiment.

5.6.1 Accuracy of Approximate Query Processing

The quality of a sample is measured by the accuracy of the approximate answers using the sample. We introduce MASP queries AQ1, AQ2 and B1. AQ1 is a relatively complex, realistic query
**Query B 1** MASG query to Bikes table.

```
SELECT from_station_id,
       AVG(age) agg1, AVG(trip_duration) agg2
FROM Bikes
WHERE age > 0
GROUP BY from_station_id
```

Figure 5.1: Maximum error for MASG query AQ1 and SASG query AQ3 using a 1% sample.

computing the changes of both the average and the number of days with high level of black carbon (bc), for each country between 2017 and 2018. The query contains different aggregates, multiple table scans, and a join operator. AQ2 and B1 are simpler examples of MASG query, which has multiple aggregate functions sharing the group-by.

Figure 5.1 shows the maximum errors of the approximate answers of query AQ1 using a 1% sample. We report the maximum error across all answers. CVOPT shows a significant improvement over other methods. It has a maximum error of 8.8% while CS and RL have error of as much as 50%. With the same space budget, the error of Uniform can be as large as 135%, as some groups are poorly represented. Similar improvements are observed with other MASG queries. For AQ2, the maximum errors of CS, RL and CVOPT are 10.1%, 29.5% and 5.9% respectively. For B1 the maximum errors of CS, RL and CVOPT are 11.7%, 8.8% and 7.7%, respectively.

We present queries AQ3, B2 and AQ4 as case-studies for SASG query. AQ4 is a realistic example, while AQ3 and B2 are simpler examples. Figure 5.1 shows the maximum errors for AQ3 using a 1% sample. For both SASG and MASG queries, CVOPT yields the lowest error. While CVOPT has 11%
Query AQ 3 Average value.

```sql
SELECT country, parameter, unit, AVG(value)
FROM OpenAQ
WHERE HOUR(local_time) BETWEEN 0 AND 24
GROUP BY country, parameter, unit
```

Query B 2 Average trip duration from each station.

```sql
SELECT from_station_id, AVG(trip_duration)
FROM Bikes
WHERE trip_duration > 0
GROUP BY from_station_id
```

Sample error, CS and RL have large errors of more than 50%. For SASG query, e.g., AQ4, CVOPT and RL share similar objective functions. However, RL assumes that the size of a group is always large, and in allocation sample sizes, does not take the group size into account (it only uses the CV of elements in the group). However real data, including the OpenAQ dataset, may contain small groups, where RL may allocate a sample size greater than the group size. CVOPT does not make such an assumption, and hence often leads to better quality samples than RL, even for the case of SASG.

Uniform has largest error of 100%, as some groups are absent in Uniform sample. Similar results are seen in other SASG queries, where CS, RL and CVOPT have the maximum errors 39%, 22% and 21% for B2; and 14%, 34% and 8% for AQ4, respectively.

Query AQ 4 Average carbon monoxide.

```sql
SELECT AVG(value),
       country,
       CONCAT(month, '_', year)
FROM (SELECT value,
       MONTH(local_time) AS month,
       YEAR(local_time) AS year,
       country
       FROM OpenAQ
       WHERE parameter = 'co' )
GROUP BY country, month, year
```
Table 5.4: Percentage average error for different queries, OpenAQ and Bikes datasets, with 1% and 5% samples, respectively.

<table>
<thead>
<tr>
<th></th>
<th>OpenAQ</th>
<th></th>
<th>Bikes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SASG</td>
<td>MASG</td>
<td>SAMG</td>
<td>MAMG</td>
</tr>
<tr>
<td>SASG</td>
<td>21.2</td>
<td>19.0</td>
<td>12.3</td>
<td>10.9</td>
</tr>
<tr>
<td>MASG</td>
<td>38.4</td>
<td>20.9</td>
<td>34.1</td>
<td>33.2</td>
</tr>
<tr>
<td>SAMG</td>
<td>2.1</td>
<td>1.1</td>
<td>3.2</td>
<td>2.3</td>
</tr>
<tr>
<td>MAMG</td>
<td>3.0</td>
<td>1.8</td>
<td>4.5</td>
<td>3.6</td>
</tr>
<tr>
<td>SASG</td>
<td>1.6</td>
<td>0.8</td>
<td>2.4</td>
<td>2.2</td>
</tr>
</tbody>
</table>

Figure 5.2: Average errors, 1% CV0PT sample answers query AQ2 with weight settings.

Table 5.4 summarizes the average errors of different queries. Generally, CV0PT shows the best average error among different methods. In some cases, CV0PT has minor improvements for average error, but for maximum error, CV0PT significantly outperforms others. That is because CV0PT gives a good representation for all groups while others do not guarantee to cover all groups. The order of other methods changes for different queries, as each method has an advantageous query type, while CV0PT is fairly stable across multiple types of query. Note that Queries AQ3, B1, and B2 have selection predicates, which are applied after the sampling is performed. We study how the sample can be reused for predicates with different selectivity and for different group-by attributes in Section 5.6.3. Since the error of Sample+Seek can be very large (maximum error as high as 173%), we exclude it from comparisons in the rest of this section.
5.6.2 Weighted Aggregates

When multiple aggregates are involved, they are not necessarily equally important to the user. CVOPT allows the user to assign a greater weight to one aggregate over the others, leading to an improved quality for the answer, based on the user’s need. We conducted experiments with query AQ2 and query B1. Each query has two aggregations, $\text{Agg1}$ and $\text{Agg2}$, with the weights denoted by $w_1$ and $w_2$, respectively. We use CVOPT to draw 3 samples with different weighting profiles: $(w_1, w_2) = \{(0.1, 0.9), (0.5, 0.5), (0.9, 0.1)\}$, as the user favors $\text{Agg2}$ over $\text{Agg1}$ in the first case and favors $\text{Agg1}$ in the third case. The second case is equal to default, non-weighted setting. Results are presented in Figure 5.2 and Figure 5.3. From the left to the right side, as $w_1$ increases and $w_2$ decreases, the average error of $\text{Agg1}$ decreases and of $\text{Agg2}$ increases. The results in both queries show CVOPT’s ability to create a sample that better fits the user’s priority. While previous heuristic works cannot systematically support weighted aggregates, we find this feature is practically useful in calibrating the sample.

5.6.3 Sample’s Usability

**On Sample Rate:** It is well known that a higher sample rate generally improves the sample’s accuracy in query processing. Nevertheless, we compare the accuracy of the different methods for queries AQ2 and B2 under different sample rates. Figure 5.4 and Figure 5.5 show that CVOPT outperforms its peers at nearly all sample rates of the study, on different datasets.
Figure 5.4: Sensitivity of maximum error to sample size. MASG query AQ2 answered by samples with various sample rates.

Figure 5.5: Sensitivity of maximum error to sample size. SASG query B2 answered by samples with various sample rates.

On Predicate Selectivity and Group-by Attributes: Queries commonly come with selection predicates, i.e., the WHERE clause in a SQL statement. Since samples are pre-constructed, the same sample is used no matter what the selection predicate is. We study the sample’s usability on the selectivity of the predicate. Query AQ3 has a predicate: WHERE HOUR(local_time) BETWEEN <low> AND <high>. By changing the filter conditions, we generated 4 queries AQ3.a, AQ3.b, AQ3.c and AQ3, with selectivity of 25%, 50%, 75% and 100%, respectively. All queries are answered by the materialized sample optimized for query AQ3.

Similarly, we have queries B2.a, B2.b, B2.c, and B2, with controllable predicate parameters to table Bikes. Figure 5.6 and Figure 5.7 show the results. Although the samples are able to serve different queries, we observe the effect of selectivity upon the accuracy. Overall, the greater the selectivity, the lesser is the error due to sampling. For each predicate query, CVOPT has a lower
Figure 5.6: Maximum error of SASS queries with different predicates on OpenAQ, answered by one materialized sample, showing the reusability of the sample.

Figure 5.7: Maximum error of SASS queries with different predicates on Bikes, answered by one materialized sample, showing the reusability of the sample.
	error than CS and RL. Table 5.5 shows the average error of a group of six different queries using the materialized sample optimized for AQ3. Note that AQ5 and AQ6 use different WHERE clauses that are also different from those used by AQ3 and AQ3.*. AQ6 also uses a different set of GROUP-BY attributes. For all six queries, CVOPT performs well with good accuracy and is consistently better than other methods.

**Query AQ 5** Average measurement for each parameter of the countries in northern hemisphere.

```sql
SELECT country, parameter, unit,
      AVG(value) average
FROM OpenAQ
WHERE latitude > 0
GROUP BY country, parameter, unit
```
Query AQ 6 Count the number of times the measurement of each parameter is higher than 0.5 in Vietnam.

```sql
SELECT parameter, unit,
COUNT(IF(value > 0.5, 1, 0)) count
FROM {input_table}
WHERE country = "VN"
GROUP BY parameter, unit
```

Table 5.5: Average error of multiple queries answered by one materialized sample, showing the reusability of the sample.

<table>
<thead>
<tr>
<th></th>
<th>AQ3</th>
<th>AQ3.a</th>
<th>AQ3.b</th>
<th>AQ3.c</th>
<th>AQ5</th>
<th>AQ6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>98.4</td>
<td>21.0</td>
<td>21.4</td>
<td>18.0</td>
<td>99.6</td>
<td>100.0</td>
</tr>
<tr>
<td>CS</td>
<td>2.5</td>
<td>5.8</td>
<td>2.9</td>
<td>2.8</td>
<td>3.9</td>
<td>0.9</td>
</tr>
<tr>
<td>RL</td>
<td>5.4</td>
<td>9.5</td>
<td>6.9</td>
<td>5.6</td>
<td>4.3</td>
<td>3.5</td>
</tr>
<tr>
<td>CVOPT</td>
<td>1.5</td>
<td>4.4</td>
<td>2.4</td>
<td>1.9</td>
<td>2.3</td>
<td>0.8</td>
</tr>
</tbody>
</table>

5.6.4 Multiple Group-by Query

WITH CUBE is an extension of group-by clause in SQL that generates multiple grouping sets of given attributes in a single query. For example, `GROUP BY A, B WITH CUBE` will generate grouping sets of (A, B), (A), (B), and (). WITH CUBE is powerful in helping user to easily and efficiently compute aggregations with a large number of combinations of group-by. We conduct experiments with SAMG queries AQ7 and B3 and MAMG queries AQ8 and B4. The queries have the grouping sets of two attributes for multiple group-by. OpenAQ has 38 countries and 7 parameters, so cube with these two attributes will generate up to 312 groups. Bikes has 619 stations and 3 years of collection, and therefore cube with `from_station_id` and `year` leads up to 2480 groups.

Query AQ 7 Single aggregate, multiple group-by, OpenAQ.

```sql
SELECT country, parameter, SUM(value)
FROM OpenAQ
GROUP BY country, parameter WITH CUBE
```
All methods RL, CS, and CVOPT, can sample in the presence of multiple groupings; for CS this is the scaled congressional sampling method and for RL it is the hierarchical partitioning method. Both CS and RL adopt a heuristic approach, which we implemented as described in their algorithm. Accuracies of different samplers are shown in Figure 5.8. We note that CVOPT performs significantly better than Uniform and RL and is consistently better than CS.

5.6.5 CPU Time

We measure the CPU time cost for sample precomputation and query processing, as the sum of the CPU cost at all nodes in a Hadoop cluster we used. The cluster includes a master node and 3 slave nodes. Each node has 2.70 GHz x4 CPU with 10 GB memory. For timing purposes, we generate a 1 TB dataset, OpenAQ-25x, by duplicating OpenAQ 25 times.
Figure 5.8: Maximum error of CUBE group-by queries.

Table 5.6: The sum of the CPU time (in seconds) at all nodes for sample precomputing and query processing with 1% sample for query AQ1 over OpenAQ and OpenAQ-25x.

<table>
<thead>
<tr>
<th></th>
<th>OpenAQ (40 GB)</th>
<th>OpenAQ-25x (1 TB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precomputed</td>
<td>Query</td>
</tr>
<tr>
<td>Full Data</td>
<td>-</td>
<td>2881.68</td>
</tr>
<tr>
<td>Uniform</td>
<td>913.95</td>
<td>40.31</td>
</tr>
<tr>
<td>Sample+Seek</td>
<td>2309.99</td>
<td>44.92</td>
</tr>
<tr>
<td>CS</td>
<td>3854.61</td>
<td>55.87</td>
</tr>
<tr>
<td>RL</td>
<td>4311.36</td>
<td>54.43</td>
</tr>
<tr>
<td>CVOPT</td>
<td>4263.07</td>
<td>59.99</td>
</tr>
</tbody>
</table>

5.6.6 Experiments with CVOPT-INF

Our experiments show that CVOPT, which optimizes for the $\ell_2$ norm of the CVs, leads to smaller errors at higher percentiles than CS and RL. We now consider CVOPT-INF, which minimizes for the maximum of CVs of all estimates ($\ell_\infty$ norm). Our results on the accuracy of CVOPT-INF on queries AQ3 and B2 are shown in Figure 5.9. Consistent with theoretical predictions, CVOPT-INF has a lower maximum error than CVOPT on both queries. At the same time, CVOPT-INF has a worse error than CVOPT at the 90th percentile and below. Overall, this shows that CVOPT-INF can be considered when the user is particularly sensitive to the maximum error across all groups. Otherwise, CVOPT with $\ell_2$ optimization provides robust estimate for a large fraction of the groups, with a small error across a wide range of percentiles.
5.7 Conclusion

We presented CVOPT, a framework for sampling from a database to accurately answer group-by queries with a provable guarantee on the quality of allocation, measured in terms of the $\ell_2$ (or $\ell_\infty$) norm of the coefficient of variation of the different per-group estimates. Our framework is quite general, and can be applied anytime it is needed to optimize jointly for multiple estimates. Choosing the $\ell_2$ of the CVs, larger errors are penalized heavily, which leads to an allocation where errors of different groups are concentrated around the mean. Choosing the $\ell_\infty$ of the CVs leads to a lower value of the maximum error than the $\ell_2$, at the cost of a slightly larger mean and median error. There are many avenues for further research, including (1) incorporating joins into the sampling framework (2) exploring $\ell_p$ norms for values of $p$ other than 2, $\infty$, (3) handling streaming data.
CHAPTER 6. SAMPLING OVER QUERY PIPELINE IN DATA WAREHOUSE

Modern big data analyses routinely involve complex pipelines in which multiple tasks are choreographed to execute queries over their inputs and write the results into their outputs (which, in turn, may be used as inputs for other tasks) in a synchronized dance of gradual data refinement until the final insight is calculated. Approximate Query Processing (AQP) offers a way of computing approximate but good enough query results quickly and cheaply. Unfortunately, the state of the art single-query AQP techniques do not easily generalize to multi-query pipelines. In a pipeline, approximate results are fed into downstream queries, unlike in a single query. Thus, we see both aggregate computation over sampled input and over approximate input.

This paper proposes a sampling-based approximate pipeline processing algorithm that uses unbiased estimation and calculates the confidence interval for produced approximate results. The key insight of the algorithm calls for enriching the output of queries with additional information. This enables the algorithm to piggyback on the modular structure of the pipeline without having to perform any global rewrites, i.e., no extra query or table is added into the pipeline. Compared to the bootstrap method, the approach described in this paper provides the confidence interval while computing aggregation estimates only once and avoids the need for maintaining intermediary aggregation distributions. Our empirical study on public and private datasets shows the high accuracy of the computed estimation, with an average error as low as 2%, using only 1% sample. It also shows the usefulness of the confidence interval. At the confidence level of 95%, the computed CI is as tight as ±8%, while the actual values fall within the CI boundary from 70.49% to 95.15% of times.
6.1 The Pipeline of Queries

Data pipelines provide a mechanism for structuring complex data processing applications into an efficient and modular composition of tables and queries. With each query reading a number of input tables and producing output tables, a pipeline can be viewed as a directed acyclic graph (DAG) in which tables are nodes and queries are edges Jacques-Silva and Zhang (2018). The pipeline is executed following the DAG topological order. Figure 6.1 shows an example pipeline with four queries and six tables, consuming data from a logging system and outputting the result to a dashboard. At each step, the result of a query is materialized as an intermediate table, which, in turn, can be used by downstream queries as input. Arranging queries into pipelines is beneficial because it yields smaller more readable queries and minimizes the impact of failures: if a single query fails it can be retried independently of other queries. Furthermore, pipelines encourage data reuse: an intermediate table can be fed to multiple downstream queries within a pipeline and can be reused across pipelines.

This work expands the sampling AQP approach that works well in the context of single queries Acharya et al. (1999a); Chaudhuri et al. (2007); Nguyen et al. (2019); Chaudhuri et al. (2017) to pipelines. When applying sampling to pipelines we must account for the impact of randomness introduced in sampled tables to the downstream queries. Specifically, we propose to use unbiased estimation and its confidence interval (CI) for the aggregation queries in sampled pipelines. The unbiased estimation and CI of an estimate $x$ are written as follows:

$$x \in [\bar{x} - \delta; \bar{x} + \delta]$$

* e.g., the result of $x = 100 \pm 5$, in which $\bar{x} = 100$ and $\delta = 5$ are the unbiased estimation and its CI respectively, is more informative than $x \approx 100$. The CI can be used to visualize the error bar and to enable a data-driven selection of the sampling rate.

While computing CIs for single queries has been well studied Chaudhuri et al. (2007); Hellerstein et al. (1997); Hu et al. (2009); Wu et al. (2010), extending it to pipelines requires additional work: we must analyze combinations of layered estimations and account for the impact of nested queries,
joins, and intermediate tables. Furthermore, since the largest volume of data is typically handled in the most upstream queries in pipelines (and then reduced by filters and aggregation), sampling is normally applied early upstream. This causes randomness in data introduced by sampling to travel longer paths in the pipeline DAG. All this makes sampling and CI computation in pipelines challenging.

The bootstrap method Agarwal et al. (2014); Zeng et al. (2014); Pol and Jermaine (2005) is commonly used to calculate a CI. While it can be adapted to pipelines, it will require high resource consumption in the pipeline setting—due to multiple invocations of the resampling procedure to materialize several possible intermediate distributions—and hence defeats the purpose of sampling.

**Problem statement:** We address the problem of sampling-based approximate pipeline processing by introducing a computation model that derives the unbiased estimation and its confidence interval for each aggregation in the pipeline. The model supports commonly used operators and aggregate functions: \texttt{SUM}, \texttt{COUNT} and \texttt{AVG}.

Our approach centers around augmenting each pipeline query to output extra information which will be sufficient for computing the unbiased estimations and CIs in each of its downstream queries. The extra computation is small enough not to degrade the asymptotic complexity of the rewritten query. To address complex dependencies among pipeline queries, we reduce the pipeline DAG to a collection of single-chained query sequences in which each step except the first one has exactly one dependency. We classify the expressions in the query sequences into two kinds, \textit{terminal} and \textit{nonterminal}, and define different estimation and CI computation formulation for them.
To the best of our knowledge, this is the first study evaluating approximate pipeline processing. Specifically, the contributions of this work are:

- We introduce a model of sampled pipelines that enables us to save resources while computing approximate pipeline results bounded by accurate confidence intervals.

- We derive unbiased estimations for commonly used operators and aggregate functions, including \texttt{SUM}, \texttt{COUNT}, and \texttt{AVG}. While some expressions are standard computations (\textit{e.g.}, \texttt{ADD}, \texttt{MINUS} operators, \texttt{SUM} aggregate), we provide an innovative approach for other nontrivial functions, \textit{e.g.}, \texttt{MULTIPLE}, \texttt{DIVIDE} operators, \texttt{COUNT}, \texttt{AVG} aggregates.

- We provide a query-DAG-aware computation model for the \textit{confidence interval}.

- Finally, we conduct an empirical study with pipelines operating on public and private real-world datasets. The study shows that the approximate answers are reasonably close to the actual answer (\textit{e.g.}, 1% sample yields less than 10% error on average) and that the CI computation is tight and highly accurate. For the confidence level of 95%, the CIs range from $\pm 8\%$ to $\pm 35\%$ for different aggregates. The actual values fall within the CI boundary from 70.49\% to 95.15\% of times.

### 6.2 Approximate Query Processing on Data Pipelines

A data pipeline is a DAG of tables (nodes) and queries (edges). Sampling some tables in a pipeline introduces randomness which then travels through downstream queries to downstream tables. To be able to compute reasonable estimated results and their CI, we must understand how randomness flows through queries, and to simplify this we linearize pipelines into straight-line chains of queries. To explain the process of linearization, let us first define two patterns of pipeline fragments in which queries have multiple inputs or multiple outputs.
Definition 3. A **join pattern** is a pipeline fragment with a query reading multiple input tables and producing a single output table.

In this work, we focus on the join pattern involving the SQL join operator ISO (1987). Figure 6.2a shows a join of tables A and B yielding table C as a result. The problem of supporting joins in approximate query processing has been investigated in prior research Chaudhuri et al. (1999); Acharya et al. (1999b); Chen and Yi (2017); Haas et al. (1994). Sampling with joins is challenging because the join operator is not distributive in general (a sample of the join of two full tables is not equal to the join of two sampled tables.) To work around this, we only support joins with the following restrictions: we allow inner joins with at most one input table sampled, and left outer joins and right outer joins with only the left and the right tables sampled respectively. A join of multiple tables is reduced to a sequence of multiple binary joins—each subject to the mentioned restrictions. If these conditions are met, the join operator becomes distributive.

A binary join can be viewed as an extension of the first table’s dimensions by combining the second table into it. Since sampled data arrives from one side of the join only, sampling the result of a join of two full tables is equal to the result of a join of the sampled input with the other table. Thus, when analyzing how randomness flows through the pipeline, we can disregard the unsampled branch of the join. In figure 6.2a, for example, we can assume that A is sampled (or carrying approximations) while B is unsampled.
Our model can also support joins with sampling applied to both inputs when the tables are sampled with a hash sampling algorithm with the hash function applied to the join columns on both sides. The result of such join is equivalent to a sample drawn from the join of two full tables. This sampling technique is called universe sampling Chaudhuri et al. (2017).

6.2.2 Divide Pattern

Definition 4. The divide pattern is a pipeline fragment in which a table is used as an input in multiple queries.

Figure 6.2b shows an example of a divide pattern: table $D$ is the input to two queries $a$ and $b$, which, in turn, produce tables $E$ and $F$, respectively. We deal with divide patterns by splitting them into two sub-pipelines with all the upstream queries and tables repeated in both. For example, the pipeline in Figure 6.2b can be split into $D \xrightarrow{a} E$ and $D \xrightarrow{b} F$.

6.2.3 Single-chained Data Pipeline

Having handled join and divide patterns as described above, any pipeline can be equivalently represented as a collection of straight-line sub-pipelines, each of the following shape:

$$T_0 \xrightarrow{Q_1} T_1 \xrightarrow{Q_2} \ldots \xrightarrow{Q_k} T_k$$ (6.1)

Let us assume that sampling is applied at the very first table $T_0$. If, instead, sampling is applied further downstream, the unsampled part of the pipeline can be disregarded since it does not introduce randomness. With this assumption, $Q_1$ can be analyzed using the traditional single-query sampling method in which the unbiased estimations and their CIs can be computed directly from $T_0$. A different approach must be used for downstream queries since they will be consuming data from tables with approximate data. First, let us define two types of aggregation expressions: terminal and non-terminal.

Definition 5. A terminal aggregation expression is an expression whose variance can be computed from its direct input, without any extra information carried from the previous step.
Definition 6. A nonterminal aggregation expression is an expression whose variance computation requires extra information in addition to its direct input.

Terminal expressions typically involve aggregation functions whose input comes directly from a sampled table or a projection of a sampled table. Computing unbiased estimations and their CIs for such expressions is well studied Cormode et al. (2011); Billingsley (2013); Van den Broeck et al. (2017); Dekking et al. (2005). Conversely, the source of randomness in nonterminal expressions comes from an approximate input table produced by upstream aggregation. I.e. the input of a nonterminal aggregation function is a random variable, and reasoning about such aggregation functions typically requires extra information.

In pipeline (6.1), where $T_0$ is sampled at the selection rate of $p$, let query $Q_1$ be a group-by query that has an aggregate $y = SUM(x)$. That aggregation is a terminal expression. The estimation $\bar{y} = \frac{1}{p} \sum x$ and its 95% CI $\delta(\bar{y}) = \pm 1.96 \sqrt{\frac{1-p}{p} \sum x^2}$ can be computed directly from table $T_0$. (Details of the computations are presented in Section 6.3.2.) Meanwhile, let query $Q_2$ contain an aggregate $z = COUNT(y)$. This is a nonterminal expression with the following estimation and 95% CI:

$$\bar{z} = C^{Q_2}(y) + \sum_{i \in G_1} (1-p)C^{Q_1}_i(x)$$

$$\delta(\bar{z}) = \pm 1.96 \sqrt{\sum_{i \in G_1} (1-p)C^{Q_1}_i(x) - \sum_{i \in G_1} (1-p)^2C^{Q_1}_i(x)}$$

Where $C^{Q_2}(y)$ is the result of aggregate $COUNT(y)$ in query $Q_2$, and $G_1$ and $C^{Q_1}_i(x)$ respectively are the set of groups in $Q_1$ and the result of aggregate $COUNT(x)$ with respect to group $i$ in query $Q_1$. (Details of the computations are presented in Section 6.3.3.) Note that $C^{Q_1}_i(x)$ cannot be computed directly from the contents of $T_1$; to enable this computation, we must add the aggregate $COUNT(x)$ into $Q_1$ and augment $T_1$’s schema with an extra column to hold this data.

A sampled table is a subset of the base table selected by a random sampling algorithm. The sampling process can be modeled statistically as follows. Let $t_j$ and $X_j = \{0,1\}$ respectively denote the $j^{th}$ row in the base table and the random variable controlling whether a row is included in the sample. Row $t_j$ is included in the sample if $X_j = 0$ and excluded if $X_j = 1$. The behavior
of all such $X_j$ variables determines the behavior of the sampling algorithm. An estimator $Y$, then, is a function $F$ parameterized by all of the data elements and all of the random variables: $Y = F((t_1, X_1), (t_2, X_2), \ldots, (t_n, X_n))$. The unbiased estimated value of the query is the expected value of $Y$ written as $E[Y]$. The variance is $\sigma^2(Y) = E[(Y - E[Y])^2] = E[Y^2] - E^2[Y]$.

6.3 Computation of the Unbiased Estimations and Confidence Interval

In this section, we introduce the calculation of the unbiased estimation and its CI for commonly used aggregations, SUM, COUNT, AVG, and operators such as additions, subtractions, products, and ratios expressions. In the single-chained pipeline described in Equation (6.1), the uniform sample is applied to table $T_0$ with a sampling rate of $p$. Our first goal is to compute the unbiased estimate for each aggregate, which best approximates the true result - if calculated from the full base table. Secondly, we provide CI formulae, that can be used to calculate a CI for every aggregate value in the $T_k$ table, along with detailed extra information that is computed at each stage of the pipeline, to support the CI computation at $T_k$.

We use $S^*$, $C^*$, and $A^*$ to represent the true value of SUM, COUNT, and AVG aggregate results, respectively. We use $S$, $C$, and $A$ to represent their estimated values, respectively. For other expressions, we use $Y$ to represent an estimator for some aggregate function or operator. For calculating a SUM aggregate somewhere in the pipeline, let the relevant records in the base table be $(a_1, b_1), \ldots, (a_C, b_C)$. We have:

$$S_a = \sum_{i=1}^{C} a_i, \quad S_{ab} = \sum_{i=1}^{C} a_i b_i, \quad S_{a^2} = \sum_{i=1}^{C} a_i^2$$

Similar notations are used for the AVG aggregate:

$$A_a = \sum_{i=1}^{C} \frac{a_i}{C}, \quad A_{ab} = \sum_{i=1}^{C} \frac{a_i b_i}{C}, \quad A_{a^2} = \sum_{i=1}^{C} \frac{a_i^2}{C}$$

6.3.1 Reducing the Computation of Confidence Interval to the Variance

Using Central limit theorem Billingsley (2013), we first assume that the error of the estimator $Y$ is normally distributed. That is, we assume that $Y$ can be modeled as: $Y \approx Q + N(0, \sigma^2(Y))$,
where \( Q \) is the actual value, and \( N = N(0, \sigma^2(Y)) \) is a normally-distributed random variable with mean zero and a variance of \( \sigma^2(Y) \). Then, if we choose numbers \( lo \) and \( hi \) so that:

\[
p = \int_{lo}^{hi} f_N(x)dx
\]

where \( f_N \) is the probability density function of \( N \), we know that there is a \( p \times 100\% \) chance that \( lo \leq Q - Y \leq hi \), due to the fact that \( Q - Y = -N \) has the same distribution as \( N \) because the normal distribution is symmetric about the origin. Then it holds that if we let two random variables \( l = Y + lo \) and \( h = Y + hi \), there is a \( p \times 100\% \) chance that the random interval \([l, h]\) contains \( Q \). For example, since \( p = \int_{-1.96\sigma(Y)}^{1.96\sigma(Y)} f_N(x)dx = 0.95 \) if we assume normality of the error, then for unbiased \( Y \) we are justified in saying that there is around a 95% chance that \( Q \) is within \([Y - 1.96\sigma(Y), Y + 1.96\sigma(Y)]\). Practically, we approximate \( \sigma^2(Y) \) with a sample-based estimate \( \bar{\sigma}^2(Y) \).

Although there is nevermore any guarantee that the error of \( Y \) is normally distributed, the statistical justification for assuming normality is typically the central limit theorem, which states that as the number of independent samples taken from a distribution approaches infinity, the observed difference between the mean of the distribution and the mean of the samples looks increasingly like a sample from a normally distributed random variable. For most of the estimators one would encounter in a data analysis environment, normality is a safe assumption. This seems to be true even when the samples are correlated. The robustness of the normality assumption stems from statistical theory [Hall and Heyde (2014)], which asserts that variants of the central limit theorem hold in generality.

The CI with a confidence level of 95% is commonly used. However, the confidence level can be treated as a parameter. The general form of the CI is \([Y - z_l\sigma(Y), Y + z_l\sigma(Y)]\), where \( z_l \) is the quantile of the standard normal distribution, at the level \( l \). The common used value of \( z_l \) is in table 6.1. By choosing the confidence level of \( l \), one can control the range of the CI boundary. The higher confidence level leads to a large quantile value of \( z_l \), thus leads to a larger CI boundary, and vice versa.
Table 6.1: The typical confidence levels and the corresponding value of $z_l$, the quantile of the standard normal distribution. By changing parameter $z_l$, we can set the confidence level $l$.

<table>
<thead>
<tr>
<th>$l$</th>
<th>$z_l$</th>
<th>$l$</th>
<th>$z_l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>1.281551565545</td>
<td>0.999</td>
<td>3.290526731492</td>
</tr>
<tr>
<td>0.90</td>
<td>1.644853626951</td>
<td>0.9999</td>
<td>3.890591886413</td>
</tr>
<tr>
<td><strong>0.95</strong></td>
<td><strong>1.959963984540</strong></td>
<td><strong>0.99999</strong></td>
<td><strong>4.417173413469</strong></td>
</tr>
<tr>
<td>0.98</td>
<td>2.326347874041</td>
<td>0.999999</td>
<td>4.891638475699</td>
</tr>
<tr>
<td>0.99</td>
<td>2.575829303549</td>
<td>0.9999999</td>
<td>5.32672886384</td>
</tr>
<tr>
<td>0.995</td>
<td>2.807033768344</td>
<td>0.99999999</td>
<td>5.730728868236</td>
</tr>
<tr>
<td>0.998</td>
<td>3.090232306168</td>
<td>0.999999999</td>
<td>6.109410204869</td>
</tr>
</tbody>
</table>

Table 6.2: Summary of the unbiased estimation computations. Details are in sections 6.3.2 and 6.3.3.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y = X \pm \alpha$</td>
<td>$Y = X \pm \alpha$</td>
</tr>
<tr>
<td>$Y = \alpha X$</td>
<td>$Y = \alpha X$</td>
</tr>
<tr>
<td>$Y = X_1 \pm X_2$</td>
<td>$Y = X_1 \pm X_2$</td>
</tr>
<tr>
<td>$S_X = SUM(X)$</td>
<td>$S_X = \sum X$</td>
</tr>
<tr>
<td>$C = COUNT(*)$</td>
<td>$C = \sum 1$</td>
</tr>
<tr>
<td>$A_X = AVG(X)$</td>
<td>$A_X = AVG(X)$</td>
</tr>
</tbody>
</table>

Applying the computation to the SUM aggregate, with 95% probability the true value $S$ lies in a confidence interval:

$$[S^* - 1.96\sigma(Y_S), S^* + 1.96\sigma(Y_S)]$$  \hspace{1cm} (6.2)

Thus it is sufficient to compute the CI once we have the variance $\sigma^2(Y_S)$. Similarly, for other SUM-related aggregate functions, the CI computation reduces to the variance computation.

We derive the formulation to compute the estimation and variance of commonly used functions and operators. Table 6.2 and table 6.3 summarize the formulations of the typical expressions. Other expressions, such as addition, subtraction, multiplication, and dividend of two SUMs aggregates,
Table 6.3: The summary of the variance computations. Details are in sections 6.3.2 and 6.3.3.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y = X \pm \alpha$</td>
<td>$\sigma^2(Y) = \sigma^2(X)$</td>
</tr>
<tr>
<td>$Y = \alpha X$</td>
<td>$0$</td>
</tr>
<tr>
<td>$Y = X_1 \pm X_2$</td>
<td>$0$</td>
</tr>
<tr>
<td>$S_X = \text{SUM}(X)$</td>
<td>$\sigma^2(S_X) = \frac{1-p}{p} \sum X^2$</td>
</tr>
<tr>
<td>$C = \text{COUNT}()$</td>
<td>$\sigma^2(C) = \frac{1-p}{p} C$</td>
</tr>
<tr>
<td>$A_X = \text{AVG}(X)$</td>
<td>$\sigma^2(Y_{A_X}) = \frac{1-p}{p} \left( S^2_{A_X} - \frac{S^2_{A}}{C^3} \right)$</td>
</tr>
<tr>
<td>$Y = S_A \pm S_B$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$Y = S_A * S_B$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

are available. Some formulations are trivial and standard computation, while other calculations are more complicated. Note that the formulations are derived under certain assumptions. For example, as the typical query in a pipeline is a group-by query, we assume that aggregate from different groups are independent. We provide the detailed proofs for the nontrivial formulation for terminal expressions in section 6.3.2 and for nonterminal expression in section 6.3.3.

### 6.3.2 Computation of the Variance for Terminal Expression

We first calculate the covariance of two \texttt{SUM}s, which are needed to calculate variance.

**Lemma 10.** Let $Y_{Sa}$ and $Y_{Sb}$ be estimators of $S_A$ and $S_B$, respectively, computed from a Bernoulli sample of the base table using sampling rate $p$, with scaling.

$$\text{Cov}(Y_{Sa}, Y_{Sb}) = \frac{1-p}{p} S_{ab} \quad (6.3)$$

**Proof.** Let there be $C$ independent random variables $X_i \in \{0, 1\}, i = 1..C$, each describing how many times the $i^{th}$ record appears in a sample of the base table. An estimator $Y_{Sa}$ is $Y_{Sa} = \sum_{i=1}^{C} \frac{X_i a_i}{p}$. 


For $i \neq j$, $X_i$ and $X_j$ are independent. So we get:

$$
Cov(Y_{Sa}, Y_{Sb}) = Cov\left(\sum_{i=1}^{C} \frac{X_ia_i}{p}, \sum_{j=1}^{C} \frac{X_jb_j}{p}\right)
$$

$$
= \sum_{i=1}^{C} Cov(X_i, X_j) \frac{a_ib_i}{p^2}
$$

Since $X_i \in \{0, 1\}$, we have $E[X_i^2] = E[X_i] = p$. So $Cov(X_i, X_j) = E[X_i^2, i] - E^2[X_i] = p - p^2$. Therefore:

$$
Cov(Y_{Sa}, Y_{Sb}) = \sum_{i=1}^{C} \left(p - p^2\right) \frac{a_ib_i}{p^2} = \frac{1 - p}{p} S_{ab}
$$

**Theorem 8.** The variance of terminal $SUM(a)$ is:

$$
\sigma^2(Y_{sa}) = Cov(Y_{Sa}, Y_{Sb}) = \frac{1 - p}{p} S_{a^2}
$$

**Proof.** Let $Y_{Sa}$ be an estimator for the true sum $S_a$, computed from a sample of the base table with a sampling rate $p$. Using equation 6.3, we have the variance of the sum estimate is $\sigma^2(Y_{sa}) = Cov(Y_{Sa}, Y_{Sb}) = \frac{1 - p}{p} S_{a^2}$.

**Theorem 9.** The variance of terminal $COUNT$ is:

$$
\sigma^2(Y_{C}) = \frac{1 - p}{p} C
$$

**Proof.** Let $Y_C$ be an estimator of the true count $C$. Since $COUNT$ is a special case of $SUM$, where $\forall i, a_i = 1$, the variance of $COUNT$ is $\sigma^2(Y_{C}) = \frac{1 - p}{p} C$.

Before deriving the variance of the average function $AVG$, we have a lemma:

**Lemma 11.** Let $Y_{Sa}/S_b$ be an estimator of $S_a/S_b$, computed from a sample of the base table with sampling rate $p$

$$
\sigma^2(Y_{Sa}/S_b) = \frac{1 - p}{p} \left(\frac{S_a^2}{S_{b^2}} + \frac{S_a^2 S_{b^2}}{S_b^4} - 2 \frac{S_a S_{ab}}{S_b^3}\right)
$$

**Proof.** Let there be $C$ independent random variables $X_i$ each describing how many times the $i^{th}$ record appears in a sample of the base table. Let function $g(x, y) = \frac{x}{y}$. An estimator $Y_{Sa}/S_b$ of the ratio of $SUM$s is $Y_{Sa}/S_b = \frac{Y_{Sa}}{Y_{Sb}} = g(Y_{Sa}, Y_{Sb})$. Using Delta method Doob (1935), a Fourier transform can be applied upon a function $f(x, y)$ so that it is approximated as $f(x_0, y_0) + f'_x(x_0, y_0)(x - x_0) +$
For the ratio function \( g(x, y) \), it holds that \( g'_x(x, y) = 1/y \), and \( g'_y(x, y) = -x/y^2 \).

So:

\[
Y_{S_a/S_b} \approx \frac{S_a}{S_b} + \frac{1}{S_b} (Y_{S_a} - S_a) - \frac{S_a}{S_b^2} (Y_{S_b} - S_b)
\]

Applying to the following standard statistic formulate: \( \sigma^2(aX + b) = a^2 \sigma^2(X) \); \( \sigma^2(X + Y) = \sigma^2(X) + \sigma^2(Y) + 2 \text{Cov}(X,Y) \); and \( \text{Cov}(aX + b, cX + d) = ac \text{Cov}(X,Y) \), we have:

\[
\sigma^2(Y_{S_a/S_b}) \approx \frac{1}{S_b^2} \sigma^2(Y_{S_a}) + \frac{S_a^2}{S_b^2} \sigma^2(Y_{S_b}) - 2 \frac{S_a}{S_b^2} \text{Cov}(Y_{S_a}, Y_{S_b})
= \frac{1 - p}{p} S_a^2 + \frac{S_a^2}{S_b^2} \left( \frac{1 - p}{p} S_b^2 - 2 \frac{S_a}{S_b^2} S_ab \right)
= \frac{1 - p}{p} \left( \frac{S_a^2}{S_b^2} + \frac{S_a^2 S_b^2}{S_b^4} - 2 \frac{S_a S_b}{S_b^3} \right)
\]

(6.4)

\[\text{Theorem 10.}\] Let \( Y_{A_n} \) be an estimator for \( \text{AVG}(a) \), computed from a sample with sampling rate \( p \).

The variance of the average aggregate is:

\[
\sigma^2(Y_{A_n}) = \frac{1 - p}{p} \left( \frac{S_a^2}{C^2} - \frac{S_a^2}{C^3} \right)
\]

Proof. \( Y_{A_n} \) is a special case of \( Y_{S_a/S_b} \), where \( S_b = S_b^2 = C \), and \( S_{ab} = S_a \). The corollary can be derived by plugging these values into Lemma 11.

\[\square\]

6.3.3 Computation of the Variance for Nonterminal Expression

\[\text{Theorem 11.}\] The variance of a nonterminal \( \sum S_X = \sum X_i \) is \( \sigma^2(S_X) = \sum \sigma^2(X_i) \)

Proof.

\[
\sigma^2(S_X) = \sigma^2 \left( \sum_{i=1}^{C} X_i \right) = C \sum_{i=1}^{C} \sum_{j=1}^{C} \text{Cov}(X_i, X_j)
= \sum_{i=1}^{C} \sigma^2(X_i) + 2 \sigma_{1 \leq i < j \leq C} \text{Cov}(X_i, X_j)
\]

(6.5)

With the assumption that \( X_i \) is pair-wise independent, we have \( \text{Cov}(X_i, X_j) = 0, i \neq j \). Thus \( \sigma^2(S_X) = \sum \sigma^2(X_i) \)

\[\square\]
The \textit{COUNT} function behaves differently depending on the upstream query. There are 2 cases whether the upstream query has \textit{GROUP-BY}. The predicate, \textit{i.e.}, \textit{WHERE} clause, can be viewed as additional \textit{GROUP-BY}, where some groups are ignored. A nonterminal \textit{COUNT} aggregate in the \textit{n} step is equivalent to the \textit{COUNT DISTINCT} function on the group-by columns in the \textit{n−1} step if that is a \textit{GROUP-BY} query. Otherwise, it is equal to \textit{COUNT} function in \textit{n−1} step. In the former case, Let $G_1, G_2, \ldots, G_g$ denote the groups in the upstream query. $C_i^{n−1} = |G_i|$ is the size of the group $G_i$.

\textbf{Theorem 12.} The unbiased estimation of a \textit{COUNT} expression in step \textit{n}, given that the previous step (\textit{n−1}) is a \textit{GROUP-BY} query, is $Y_{C_n} = C_n + \sum_i (1−p)^{C_i^{n−1}}$.

\textit{Proof.} As \textit{COUNT} in step \textit{n} is equivalent to \textit{COUNT DISTINCT} in step \textit{n−1}, the true value is $C_n = g$, which is the number of groups in step \textit{n−1}. Let the random variable $z_i = (0, 1), i \in 1 \ldots g$ denotes whether group $G_i$ appears in the sample. The expected value of \textit{COUNT} function in \textit{n} step is $C_n = \sum_{i=1}^{g} z_i$. A group appears in the approximation when at least one of its records is selected into the sample, thus $Pr[z_i = 1] = Pr[\exists x_j \in G_i | t_j = 1]$ Recall that $t_j$ is a random variable showing if a record is selected into the sample. $t_j = 1$ if record $x_j$ is selected. Otherwise $t_j = 0$. $Pr[t_j = 1] = p$ is the sample rate. We have

$$E[z_i] = Pr[z_i = 1] = 1 - Pr[z_i = 0] = 1 - Pr[\forall x_j \in G_i, t_j = 0]$$

$$= 1 - \prod_{j=1}^{G_i} (1 - p) = 1 - (1 - p)^{C_i^{n−1}} \tag{6.6}$$

Thus $E[C_n] = \sum_{i=1}^{g} E[z_i] = C_n^* - \sum_{i=1}^{g} (1−p)^{C_i^{n−1}}$. The expected value $E[C_n]$ of a \textit{COUNT} function in step \textit{n} is less than the actual value $C_n^*$ by $\sum_{i=1}^{g} (1−p)^{C_i^{n−1}}$. This amount can be approximated by the values observed from the sample $\sum_{G_i \in \text{Sample}} (1−p)^{C_i^{n−1}}$. We have the unbiased estimation of the \textit{COUNT} is $Y_{C_n} = C_n + \sum (1−p)^{C_i^{n−1}}$.

This result is intuitive. \textit{COUNT} result tends to be less than the actual value, as infrequent groups may be absent. In fact, approximate \textit{COUNT} is \textit{always} less than or equal to the true value. To have the unbiased estimate, the group count $C_i^{n−1}$ is needed in the \textit{(n−1)th query}. 

Theorem 13. The variance of a nonterminal COUNT expression in step $n$, given that step $n-1$ is a GROUP-BY query, is

$$
\sigma^2(C^n) = \sum_i (1 - p)^{C_i^{n-1}} - \sum_i (1 - p)^{2C_i^{n-1}}
$$

Proof. We have the variance of $z_i$

$$
\sigma^2(z_i) = E(z_i^2) - E^2(z_i)
= \left(1 - (1 - p)^{C_i^{n-1}}\right) - \left(1 - (1 - p)^{C_i^{n-1}}\right)^2
= (1 - p)^{C_i^{n-1}} - (1 - p)^{2C_i^{n-1}}
$$

Thus the variance is

$$
\sigma^2(Y_{C^n}) \approx \sigma^2(C^n) = \sigma^2\left(\sum z_i\right) = \sum \sigma^2(z_i)
= \sum_i (1 - p)^{C_i^{n-1}} - \sum_i (1 - p)^{2C_i^{n-1}}
$$

(6.7)

Theorem 14. The unbiased estimation and its variance of a COUNT expression in step $n$, given that step $(n-1)$ is not a GROUP-BY query, is $Y_{C^n} = C^n$ and $\sigma^2(C^n) = \sigma^2(C^{n-1})$.

In case the upstream query does not have GROUP-BY or predicate, i.e., WHERE clause, the COUNT aggregate is equivalent to the COUNT on the upstream query. If that query does not contain COUNT aggregate on the same column, it has to be added as extra computation. It is straightforward to compute the estimation and variance in this case.

Theorem 15. The variance of the AVG aggregate is

$$
\sigma^2(A_X) = \frac{1}{C^2} \left( \sum \sigma^2(X) + A_X^2 \sigma^2(Y_C) \right)
$$

Proof. Let $f(x, y) = \frac{x}{y}$. Applying the delta method Doob (1935), we use the first-order Taylor polynomial to approx function $f(x, y)$ at $(x_0, y_0)$ as

$$
f(x, y) \approx f(x_0, y_0) + f_x'(x_0, y_0)(x - x_0) + f_y'(x_0, y_0)(y - y_0)
$$
It holds that \( f'_x(x, y) = 1/y \), and \( f'_y(x, y) = -x/y^2 \). Thus \( f(x, y) \approx x/y_0 - x_0 * y/y_0^2 + x_0/y_0 \).

Since \( A_X = \frac{S_X}{C} \), the estimator \( Y_{AX} \) of the AVG(\( X \)) is: \( Y_{AX} = \frac{Y_{S_X}}{Y_C} = f(Y_{S_X}, Y_C) \). Let \( (x_0, y_0) = (S_X, C) \), we have \( Y_{S_X/C} \approx Y_{S_X/C} - S_X * Y_C/C^2 + S_X/C \). Then the variance is

\[
\sigma^2(A_X) = \sigma^2 \left( \frac{Y_{S_X}}{C} \right) = \sigma^2(S_X)/C^2 + \sigma^2(C) * S_X^2/C^4
\]

\[
= \frac{1}{C^2} \left( \sum \sigma^2(X) + A_X^2 \sigma^2(C) \right)
\]

**Lemma 12.** The variance of the addition and subtraction of SUMs, \( Y_{S_a \pm S_b} \), is \( \sigma^2(Y_{S_a \pm S_b}) = \frac{1 - p}{p} (S_a^2 + S_b^2 \pm 2S_{ab}) \).

**Proof.** Since \( Y_{S_a \pm S_b} = Y_{S_a} \pm Y_{S_b} \), we have:

\[
\sigma^2(Y_{S_a \pm S_b}) = \sigma^2(Y_{S_a}) + \sigma^2(Y_{S_b}) \pm 2Cov(Y_{S_a}, Y_{S_b})
\]

\[
= \frac{1 - p}{p} (S_a^2 + S_b^2 \pm 2S_{ab})
\]

**Theorem 16.** The variance of the product of SUMs is \( \sigma^2(Y_{S_a* S_b}) = \frac{1 - p}{p} (S_a^2S_b^2 + S_a^2 S_{b^2} + 2S_a S_b S_{ab}) \).

**Proof.** Using Taylor approximation,

\[
Y_{S_a* S_b} \approx S_a * S_b + S_b (Y_{S_a} - S_a) + S_a (Y_{S_b} - S_b)
\]

we have:

\[
\sigma^2(Y_{S_a* S_b}) \approx S_b^2 \sigma^2(Y_{S_a}) + S_a^2 \sigma^2(Y_{S_b}) + 2S_a S_b Cov(Y_{S_a}, Y_{S_b})
\]

\[
= S_b^2 \frac{1 - p}{p} S_a^2 + S_a^2 \frac{1 - p}{p} S_b^2 + 2S_a S_b \frac{1 - p}{p} S_{ab}
\]

\[
= \frac{1 - p}{p} (S_a^2S_b^2 + S_a^2 S_{b^2} + 2S_a S_b S_{ab})
\]
Algorithm 8: A practical backward-forward algorithm to apply sampling to a data pipeline.

1. Input: \( G = (T, Q) \) – The pipeline as a DAG of tables \( T \) and queries \( Q \)
2. Output: \( \overline{G} = (\overline{T}, \overline{Q}) \) – The rewritten pipeline
3. \( T_0 = \text{SampleTable}(T_0) \)
4. \( P = \text{TopologyPath}(G) \) # Flat out \( G \) into paths
5. for \( P = (T^P, Q^P) \) in \( P \):
   6.   for \( i = |P| \) down to 1:
      7.     \( \delta = \text{RequireInput}(T^P_i) \)
      8.     \( \Delta_{i-1}^P = \Delta_{i-1}^P \cup \delta \)
   9. for \( (T_i, Q_i) \) in \( G \):
      10. \( \overline{T_i} = T_i \cup \Delta_i \) # Update table schema
      11. \( \overline{Q_i} = Q_i \cup \Delta_i \) # Add extra comp. to query
12. return \( \overline{G} = (\overline{T}, \overline{Q}) \)

6.4 Two Passes Model for Pipeline Confidence Interval Computation

To enable a pipeline to leverage sampling with confidence interval computation, we need to understand the queries and aggregations involved in the process. To achieve that, we propose a two-pass algorithm, which first does a backward (sink to source) pass on the DAG followed by a forward pass (source to sink). This is described in more detail in Algorithm 8. The algorithm gets as input the pipeline itself – the DAG of tables and queries – and the table that we will apply sampling to. It then applies the first pass by traveling backwards in each single-chained sequence of queries. In each step, the algorithm computes the information needed to produce all estimations and their CIs. In the second pass, the algorithm combines this extra information into the original pipeline, starting from the first query. As part of this process, both queries and the schema of the involved tables are updated. The output of this process is a new DAG that has the same number of nodes and edges as the original pipeline, but with modified queries and table schemas to account for the confidence interval computations.

Recall that a single-chained sequence of \( k \) queries and \( k + 1 \) tables can be view as:

\[
T_0 \xrightarrow{Q_1} T_1 \xrightarrow{Q_2} \ldots \xrightarrow{Q_k} T_k
\]
Assume that we are interested in the computation of the CI of the aggregate $y = \text{SUM}(x)$ in the last query $Q_k$. For this instance, to compute the CI $\delta(y)$, we need the variance $\text{Var}(y)$, as described in section 6.3.2 and section 6.3.3. We have $\text{Var}(y) = \text{Var}(\sum(x)) = \sum \text{Var}(x)$. Thus we need $\text{Var}(x)$ to be available in $T_{k-1}$. Similarly, based on the aggregate function used to compute $x$ in $T_{k-1}$, we then can determine the extra information needed, if any, to be added on $Q_{k-1}$, and so on. The procedure is repeated until we reach $Q_1$. Since $T_0$ is the table we apply sampling to, we can directly compute any needed information. The first visit of the pipeline should be in backward order because the extra information needed for step $n$ is added to the step $n - 1$. Computing the extra information itself may require step $n - 2$ to provide this extra information. Thus, this step is executed in the backward order of the pipeline.

After discovering all the extra information needed in each step, the second phase adds the actual computation to the pipeline. The second pass follows the direction of the pipeline to correctly and efficiently add the extra information, as, in many cases, different aggregates share the same required extra information. Furthermore, each step is computed as a SQL query, so the column name of the extra information is needed to add this extra computation into the modified SQL query. In a brief summary, the algorithm supports pipelines containing operator and aggregations mentioned in Section 6.3 and combination of them, including algebra operators such as Add, Minus, Multiple, Division; Aggregation Sum, Count, and Average. The join is partially supported with certain constrains as mentioned in Section 6.2.

### 6.5 Empirical Study

We conducted experiments to study the usefulness of our proposed method. We introduce OpenAQ pipeline – a case study pipeline of 5 queries in a publicly available dataset, in which we can see in detail how the aggregates behave differently depending on their positions in the pipeline. We also report the effectiveness of the method in the second pipeline named PrivEnt – an analytic pipeline that do analytic on a private enterprise dataset.
Query 1 Aggregation by location. Aggregations on top of the sampled table with COUNT and AVG terminal expressions.

```
INSERT INTO Location
SELECT location, city, country, parameter, unit,
     COUNT(*) count_location,
     AVG(value) avg_value
FROM OpenAQ
GROUP BY location, city, country, parameter, unit
```

The first metric we measure is the accuracy of the estimations, which reflects on the error of the approximate answer from the sampled pipeline, compared to the baseline answer from the original pipeline. Given a group-by query (a query without group-by can be viewed as it has a single group), let $x_i$ and $\bar{x}_i$ be the baseline and the approximate results of an aggregate $X$, respect to group $i$. The error of this estimate is defined as

$$e_i = \frac{|x_i - \bar{x}_i|}{x_i}$$

Let $g$ be the number of groups of aggregates $x$, the errors form a vector $[e_1, e_2, \ldots, e_g]$.

Secondly, we show the results of CI computation, as well as its usefulness in visualization. While the sampling error shows the accuracy of the sample, the value of CI boundary shows its precision. In which the smaller CI boundary means the estimates are more concentrated around the actual value. We measure the coverage of the CI, which is the percentage of time when the true value falls within the computed CI boundaries. We compare the coverage metric to the expected confidence level to measure the CI computation quality. We also study the sensitivity of CI boundaries to the parameters sample rate and confidence level.

6.5.1 Data Pipeline on the OpenAQ Dataset

In this section, we describe a case study with a synthetic pipeline with 5 queries that use the OpenAQ public dataset OpenAQ (2019). The pipeline computes a collection of the air quality measurements of different substances, such as carbon monoxide ($CO$), sulfur dioxide ($SO_2$), etc. The input dataset consists of more than 200 million records, collected daily from ten thousand
Query 2 Aggregation by city. Aggregations on an approximate values with nonterminal expressions $\text{SUM}$ and $\text{AVG}$.

```
INSERT INTO City
SELECT city, country, parameter, unit,
    \text{SUM}(\text{count\_location}) \text{ count\_city},
    \text{AVG}(\text{avg\_value}) \text{ avg\_value}
FROM Location
WHERE unit = \$\text{\mu g/m}^3\$
GROUP BY city, country, parameter, unit
```

Query 3 Final aggregation by country. This computation needs to include the CI.

```
INSERT INTO Count\_Country
SELECT country, parameter, unit,
    \text{SUM}(\text{count\_city}) \text{ count\_country}
FROM City
GROUP BY country, parameter, unit
```

locations in 67 countries, from 2015 to 2018. The dataset is stored in a table named OpenAQ, in which we apply sampling. If not otherwise noted, the sample is uniform at 1%. The visualization of the pipeline’s DAG is in Figure 6.1 where the order of computation is left to right. First Query 1 directly queries from table OpenAQ. Consecutively we have Query 2 whom result is consumed by both Query 3 and 4. While the result of Query 3 is an output of the pipeline, Query 5 further consumes the result of Query 4 to provide another output.

Applying the proposed model, we flatten the DAG into 2 single-chain sequences of queries: (i) $P_1$ being OpenAQ, Location, City, and Count\_By\_Country, and (ii) $P_2$ being OpenAQ, Location, City, Country, and Value\_By\_Country.

Query 4 Aggregation by country. $\text{SUM}$ over approximate values makes the function a nonterminal expression.

```
INSERT INTO Country
SELECT country, parameter, unit,
    \text{SUM}(value) value
FROM City
GROUP BY country, parameter, unit
```
Query 5 Final aggregation by country. The SUM of observation values needs the CI computation.

```
INSERT INTO Value_Country
SELECT country, SUM(value) value
FROM Country
GROUP BY country
```

Figure 6.4: The DAG for PrivEnt data pipeline. The execution order is from left to right.

Next, the backward scan computes all the extra information required in each step to provide the unbiased estimates and the CIs for the aggregations at Count_By_Country and Value_By_Country. For example, in $P_1$, to support Query 3, the variance of count_city is added to table City. On the path $P_2$, to support Query 5, the variance of avg_value is required. Thus, the computations of both count_city and avg_value are added to Query 2.

6.5.2 Data Pipeline on a Private Enterprise Dataset

Our second case study is the application of the proposed model on a real pipeline, named PrivEnt, that runs daily to perform an analytic. The pipeline DAG is described in Figure 6.4 with the execution order from left to right. This pipeline involves joins a fact table with a dimension table. We apply a hash sampling algorithm to draw samples of both tables. If not otherwise specifics, the sample rate is set to 5%. The last table on the right powers a visualization and monitoring dashboard where we want to display CIs. The aggregations computed by this pipeline include SUM, COUNT and AVG. The specifics of the queries and data are omitted as their details are not relevant to the obtained results.
6.5.3 Accuracy of the Approximate Results

We compare the estimate results to the ground truth that is derived from the baseline pipeline that queries on top of the original non-sampled table. Let $x_i$ and $\bar{x}_i$ be an aggregate result and its estimation of a group $i$, the error is defined as $e_i = \frac{|x_i - \bar{x}_i|}{x_i}$. Since the query is a group-by, the error of all groups is in the form of a vector $[e_1, e_2, \ldots]$. Figure 6.5 shows the distribution of the error vector of different aggregations in PrivEnt pipeline, where the sample rate is 5%. While some outlie cases can reach above 100% error, overall, the median error is less than 10% and the third quantile of most of the aggregates are less than 20%. Take a closer look at the first attributions, that has the maximum error of 26.3% and the medium error of 4.4%. It means that, even the error may reaches 26.3% in some rows, half of the number of rows in the final table for this attribute are
Figure 6.7: The median error of the approximate results of Query 3 and Query 5, OpenAQ pipeline, at various sample rates.

no more than 4.4% different from the true value. Minor difference can be seen among attributions, as the mediums of the errors various from 3.0% to 9.4%. One of the aggregates (the last one in Figure 6.5) has relatively high error due to missing groups.

The sample quality heavily depends on the sample rate. Figure 6.6 and Figure 6.7 show the sensitivity of the error to the sample rate in the PrivEnt and the OpenAQ pipelines, respectively. To no surprise, when the sample rate increases the error decreases, in both pipelines. From our observation, on the OpenAQ pipeline, using a 1% sample, we can expect the median error of 7.2% to 8.0%. On the PrivEnt pipeline, the 5% sample gives us a median error from 3.0% and 9.4%. These errors are practically acceptable, especially for visualization purpose where the approximate results are used to power monitoring dashboard. Thus we use 5% and 1% as default sample rate for PrivEnt and OpenAQ pipelines, respectively. For more error-sensitive applications, increasing sample rate would help to mitigate the discrepancy.

Another practically important measurement is the missing results. As the input table is uniformly sampled, the underrepresentation can vanish from the sample table. Besides being counted as a 100% error in the aggregates error, the missing result leads to further inaccuracy, depending on the nature of the analysis. For example, assume that results from Query 5 in OpenAQ pipeline power a world-maps of sulfur dioxide ($SO_2$). Some countries that have a relatively fewer number of data collected may be missing on these maps. Figure 6.8 shows the percentage of missing results in the final table of the OpenAQ and PrivEnt pipelines, at different sample rates. As expected, when
Figure 6.8: Percentage of missing results in Query 3, OpenAQ pipeline and the final table of the PrivEnt pipeline, at various sample rates, comparing to the full results.

Figure 6.9: The exact and estimated results, with the 95% CI boundaries, of the estimation in Query 3. Limited to top 50 results for the display purpose.

As the sample rate increases, number of missing results decreases. When only 5% of the input data are used, we see 9.2% and 10.6% of rows in the final table are missing in OpenAQ and PrivEnt pipelines, respectively. Despite the overall low error, one may found the of missing row is high, depending on the particular application. In this work, we use uniform random sampling. Recent works on stratified random sampling have shown the better representation for infrequent data Nguyen et al. (2019, 2020); Acharya et al. (2000); Chaudhuri et al. (2007), that can potentially be incorporated into our model.
Figure 6.10: The exact and estimated results, with 95% CI boundaries, of the PrivEnt pipeline. We hid the labels of the data. Limited to top 10,000 results for the display purpose.

6.5.4 CI of the Estimations

Figure 6.9 and Figure 6.10 show the visualizations of aggregate count_country of query 3, in the OpenAQ pipeline, and an aggregate at the final step of the PrivEnt, respectively. We limit the number of results for presentation purposes. Although we only use 1% of the data to process the pipeline, in both figures, the approximate numbers are well aligned with the exact answers. Furthermore, these two figures show the 95% CI boundaries around the estimate answer. Note that the exact answer is only obtained from running the pipeline on the full dataset. On the actual usage of the sampling, the CI boundary give the visualization of the range in which the real value is highly likely at.

Most of the time, the true value falls within this boundary range, as seen in Figure 6.9 and Figure 6.10. The computed CIs are in a narrow range around the approximate results, showing that the CI values are tight, thus it provides value to users. If CIs are computed in a naive way, they can end up having a large range, which increases the chances of covering the true aggregation value. For example, the CI of $[-\infty, +\infty]$ is guaranteed to cover any estimation. Nonetheless, these types of CI are useless, as it provides no information to the user.

6.5.5 Accuracy and Confidence Level of the CI

We show the CI boundaries at a confidence level of 95% in Figures 6.9 and Figure 6.10. By changing the confidence level, we can control how tight the boundary is. Intuitively, with higher
Figure 6.11: CI coverage in Query 3, OpenAQ pipeline – percentage of true results fall within the CIs boundaries. The coverages of computed CIs, at various quantile value $z_i$, are close to the expectation numbers.

Table 6.4: The CI value and its coverage - percentage of actual results fall within the CI, of Query 3 and 5, OpenAQ pipeline, and some aggregates in PrivEnt pipeline, at different confidence levels.

<table>
<thead>
<tr>
<th>$z_i$</th>
<th>Expected Coverage</th>
<th>OpenAQ pipeline</th>
<th>PrivEnt pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Query 3</td>
<td>Query 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cov.</td>
<td>CI</td>
</tr>
<tr>
<td>1.28</td>
<td>80.00%</td>
<td>78.60%</td>
<td>±7%</td>
</tr>
<tr>
<td>1.64</td>
<td>90.00%</td>
<td>88.77%</td>
<td>±9%</td>
</tr>
<tr>
<td>1.96</td>
<td><strong>95.00%</strong></td>
<td>94.39%</td>
<td>±11%</td>
</tr>
<tr>
<td>2.32</td>
<td>98.00%</td>
<td>96.14%</td>
<td>±13%</td>
</tr>
<tr>
<td>2.57</td>
<td>99.00%</td>
<td>97.19%</td>
<td>±15%</td>
</tr>
</tbody>
</table>

Confidence, the CI boundary is larger. Thus, it is more likely that the true value falls within the CI. On the other hand, the larger CI range gives a vague estimation, thus it is less useful to the users. This section shows the study of the effect of the confidence level on the computed CIs.

The CI of estimation can be given in the form of the percentage of the estimated value, i.e., $x = \bar{x} \pm \delta\%$, where $x$, $\bar{x}$ and $\delta$ are the true answer, its estimated value, and the CI boundary, respectively. This format gives us the relative difference between the CI and its estimate, despite the magnitude value of the estimation. We use this form of CI to compare the CI boundary of different estimates without comparing the estimations themselves. For example, given two estimations $x = 5 \pm 10\%$ and $y = 100 \pm 10\%$, we can say that both of them have the same boundary range. While the absolute value of the CIs are 20 times different.
The true value *likely* falls within the CI range, however, there is no absolute guarantee that every single answer are within the computed CI boundaries. Given that we have multiple estimate answers, we use the *coverage* metric to measure the accuracy of the computed CI. This metric counts the number of times the CI covers the true value. That is, when \( x \in [\bar{x} - \delta; \bar{x} + \delta] \), we say that the CI is accurate. This *coverage* may be different than the expected confidence level. For example, given a 95% CI, the actual values are expected to falls within the CI boundaries with 95% chances. However, compared to the baseline answers, we may see 9 out of 10 estimations have their true value within the CI boundaries. We said the coverage is 90%. Note that the accuracy of the CI is only known when the true value is known.

We change the confidence level \( l \), by adjusting the quantile value \( z_l \) as given in table 6.1, to show the effect of the confidence level on the CI, specifically on its accuracy (i.e., coverage) and range. Figure 6.11 shows the coverage of the CI in query 3, OpenAQ pipeline, comparing to the confidence level. The computed CIs have the coverage closely tracking the expected numbers. It means that the CI boundaries are trustworthy. *e.g.*, we have the 95% CI boundaries that cover true values 94.4% of times; the 98% CI boundaries cover 96.1% of times.

Table 6.4 shows the sensitivity of the CI and its coverage to the confidence level. We vary the parameter \( z_l \) from 1.28 to 2.57, corresponding the CIs are expected to coverage 80% to 99%. At the commonly used confidence level of 95%, we see that 94.39% of the time, the true value of the aggregations in Query 3 falls within its CI of \( \pm 11\% \). This means that, with 94.39% chances, \( x \in [\bar{x} \pm 11\%] \). As expected, when \( z_l \) increases, the accuracy increases. However, the CI becomes larger, thus being less informative to users. For example, we see that the CI of Query 3 increases from \( \pm 7\% \), at 80% confidence, to \( \pm 11\% \) and \( \pm 15\% \), at 95% and 99% confidence. In general, the coverage increases as \( z_l \) increases. However they are different for different aggregates. In one pipeline, some aggregates are *over*-coverage, the other may be *under*-coverage, comparing to the expected number.
CHAPTER 7. CONCLUSION

The cost efficiency is one of the most important aspect in designing a data warehouse system, due to the fact that a data warehouse consumes large amount of energy in order to provide the enormous computation and storage power. Approximate query processing is a well-known approach to handle massive data, in which the random sample is used to represent the base data. Random sampling significantly reduces resource consumed and response times, at the cost of a small approximation error. For many applications, a small error is justifiable. This work focus on the approximate query processing, including algorithms to draw samples, methods to maintain sample quality, and effective usages of the sample for approximately answering different classes of queries.

First, we study sampling algorithms, that is the foundation of the approximate query processing. We propose the optimal stratified sampling algorithm that is optimized for population aggregate query. As the query involves, we improve the sampling algorithms for group-by aggregate queries. Finally, we introduce the sampling over the pipeline model of queries processing, where multiple queries and tables are involved. Complex analysis tasks routinely involve complex pipelines, such that multiple tasks are choreographed to execute queries over their inputs and write the results into their outputs. We propose a sampling-based approximate pipeline processing framework that provides both the unbiased estimation and the confidence interval for the approximate results.

Our empirical study shows that our sample for population aggregate queries are optimal. The sample for group-by queries outperforms the current state-of-the-art on sample quality and estimation accuracy. The experiments for approximate pipeline processing show the high accuracy of the computed estimation, with an average error as low as 2%, using only a 1% sample. It also shows the usefulness of the confidence interval. At the confidence level of 95%, the computed CI is as tight as $\pm 8\%$, while the actual values fall within the CI boundary from 70.49% to 95.15% of times.
BIBLIOGRAPHY


Huang, W., Zhou, Y., and Yu, B. Query pipeline.


Pfleiger, T. F., Kimball, A. E., and Desai, A. A. Distributed query engine pipeline method and system.


