Evaluating Join Query Estimation Accuracy in AQP using Bootstrap Sampling on Correlated Datasets

by

Sabin Maharjan

A thesis submitted to Youngstown State University in partial fulfillment of the requirements for the degree of

Master of Computer and Information Systems

YOUNGSTOWN STATE UNIVERSITY

Fall, 2023

Evaluating Join Query Estimation Accuracy in AQP using Bootstrap Sampling on Correlated Datasets

Sabin Maharjan

Abstract

Approximate query processing (AQP) is a promising approach for processing queries on big data. However, the accuracy of AQP is difficult to evaluate, especially for join queries. This thesis investigates the use of bootstrap sampling to assess the estimation errors of join queries processed by AQP.

A correlated sampling scheme is used to generate join-preserving samples from base relations. Bootstrap replicates derived from the correlated samples are leveraged to construct confidence intervals that quantify the estimation errors. A prototype system is implemented with Rust to realize the methodology.

Extensive experiments conducted on TPC-H benchmark datasets demonstrate the feasibility of using bootstrap sampling in conjunction with correlated sampling to evaluate the accuracy of join query estimations in large-scale AQP systems. The results show that the proposed approach can significantly improve the estimation efficiency and accuracy compared to naive sampling.

Acknowledgments

I am deeply grateful to my advisor, Dr. Feng Yu, for his patience, guidance, and support throughout this research project. His expertise, patience, and encouragement have been invaluable to me. I am also grateful to the members of my thesis evaluation committee, Dr. Alina Lazar, and Dr. John R. Sullins for their thoughtful feedback and suggestions.

I would also like to thank my family for their love and support. My parents, Sanu Kaji Maharjan and Sanu Maiya Maharjan have always been there for me, and I am so grateful for their sacrifices and encouragement. I would also like to thank my siblings and my wife for their love and support.

Finally, I would like to thank all of the other people who have helped me along the way, including my friends, colleagues, and mentors. I am truly grateful for their support.

This thesis would not have been possible without the help of all of these fantastic people. I am deeply indebted to them.

Contents

List of Figures

1 Introduction

Big data refers to large, complex data sets that are difficult to manage using traditional data processing software which encompasses the volume, velocity, and variety of information. In 2020, every human created at least 1.7MB of data per second as we created 2.5 quintillion data bytes daily[1]. With the increase of people who have internet access, it is estimated that we will generate 463 exabytes of data each day by 2025[1]. The explosive growth of big data with the rise in the number of people online has presented both opportunities and challenges for organizations seeking to extract valuable insights.

Approximate query processing (AQP) is one of the ways to tackle big data. It provides approximate answers to queries on big data in a shorter time than traditional query processing systems. This can be achieved by sampling the data and using statistical methods to estimate the results of the query. However, when working with complex join queries the AQP can be inaccurate. Join queries involve combining data from multiple tables and requires samples that have preserved join relationship between tables for accurate estimates of the query results. Simple random sampling creates samples independently for each table and joining the samples that have been created independently will lose its join relationship. Using these samples can result in inaccurate estimations.

This paper extends the work of Cal [2], who proposed an optimized bootstrap sampling for error estimations. The work done by Cal was on simple queries for a single table while this thesis addresses the complex join query error estimation. The work done by Cal[2] focused on estimating errors for simple queries where the AQP was only used on a single table. In order to address the complex join query we use correlated sampling [9] to generate join preserving samples for accurate estimations of join queries. We utilize the optimized bootstrap sampling [2] to estimate the error.

The remainder of the thesis is organized as follows: Section 2 provides relevant background. Section 3 discusses correlated sampling for join tables. Section 4 details using Bootstrap for query estimation. Section 5 presents experiments and evaluations. Finally, Section 6 concludes the thesis.

2 Background

2.1 Approximate Query Processing

Approximate Query Processing is a technique that returns approximate answers to queries in less time using fewer resources than when getting the exact answer. Executing queries on big data can be time-consuming and resource-intensive. AQP helps to address these issues and one of the key focuses of this paper is Selection AQP (σ-AQP), as explored in the work by Cal et al [2]. Selection queries are a type of query that filters rows based on certain conditions. In the context of σ-AQP, rather than executing a selection query directly on the entire dataset, a different approach is taken. A random sample without replacement (SRSWOR) is drawn from the dataset, denoted as S. Then we use the Correlated sampling technique to create samples that have a join-relationship preserved. After the samples are created we run the join query on the correlated samples to get the query result Y_s.

2.2 Correlated Sampling

Correlated Sampling is a statistical technique for generating a sample of tuples from a database that preserves the relationship between tuples in different relations. As explored by Wilson [9], it is used in addition to simple random samples without replacement to generate samples that preserve the joined relationship between tuples and their relations. The main objective of CS2 is to create an unbiased, fast, and

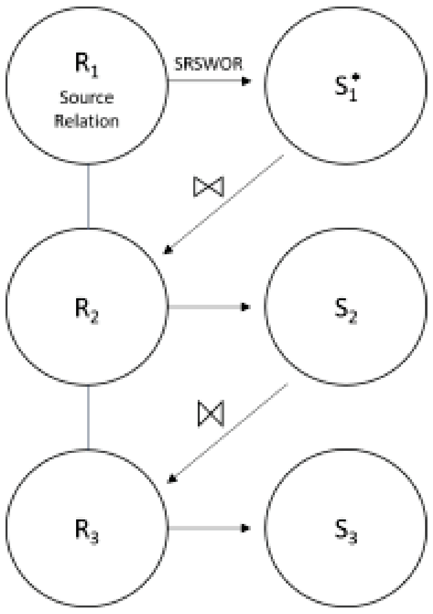


Figure 1: Correlated Sampling CS2

precise estimation for queries that involve all types of joins and selections [9]. Using only SRSWOR to create a sample and joining each sample to create join samples results in loss of join relationships. Figure 1 illustrates the process of creating a sample once the source relation and sample path are determined. Once the source relation is defined, Simple random sampling without replacement (SRSWOR) is performed on the source relation denoted as R_1 which results in a sample relation denoted as S_1^*. To create the second sample S_2, we join the S_1^* with the relation R_2 preserving the join relationship. S_2 is joined with R_3 to create S_3. The example only consists of a single edge, in case of multiple edges to multiple relations we should exhaust all the possibilities.

2.3 Bootstrap Sampling

The bootstrap sampling method is one of the widely used resampling techniques in modern statistical inference [5] introduced by Bradley Efron in 1979 [6], to measure the quality of various statistical estimators. It provides a powerful technique to estimate the properties of an estimator such as standard error, and confidence interval without requiring distributional assumptions. Bootstrap sampling or bootstrapping is the concept of pulling oneself by one's bootstrap [4]. The key concept of bootstrapping is to approximate the sampling distribution of an estimator by resampling from the original dataset. Given the original data sample n, bootstrapping involves resampling the original sample using a simple random sample with replacement of the same size n, as the original sample. The statistics of interest like mean, standard error, and confidence intervals are computed on each bootstrap sample, generating a vector of bootstrap statistics. This vector forms an approximation of the sampling distribution of the statistics [6] which is used for the estimation of error for our join selection query.

3 Correlated Sampling For Query Estimation

There are two ways for query estimations using Correlated sampling, they are source query estimations and no source query estimations. As the name suggests source query estimation is to estimate the result size which involves the initially sampled source relation S_1^*. As illustrated in fig 1 we have the source relation R_1 and then we sample it using SRSWOR to create an unbiased sample S_1^*. For example join relation between S_1^* and S_2 is considered a source query which can be used to estimate the the join query estimation between R_1 and R_2. The following formula is used to estimate the query result

$$\hat{Y}source = \frac{N_1}{n_1} \sum_{i=1}^{n_1} y_i \tag{1}$$

where $\hat{Y}source$ is the estimated result size of the source query. N_1 is the total number of tuples in the source relation R_1 and n_1 is the number of tuples from sample S_1^*. y_i is the number of results tuples produced by the i^{th} tuple in S1* when joined with the other sample relations.

No source query estimation as the name suggests is about estimating query results using samples that do not include source relation. As illustrated in fig 1 R_1 is the source relation and S_1^* is the sample from the source. A join can be considered no source when the join does not have S_1^*. For example, a join of S_2 and S_3 is a no-source query. Since there is no direct sample from the source, additional steps are required to derive unbiased result size estimates for such queries. This is achieved using a precomputed Joinable Tuple Sampled Ratio or JR that links each tuple to the source sampling rate.

3.1 Implementation

In this research we have used the dataset generated using TPC-H benchmark [8]. The benchmark provides us with 8 tables as we see in the figure 2. The first thing to do for creating correlated samples is to choose the source table. As the Lineitem

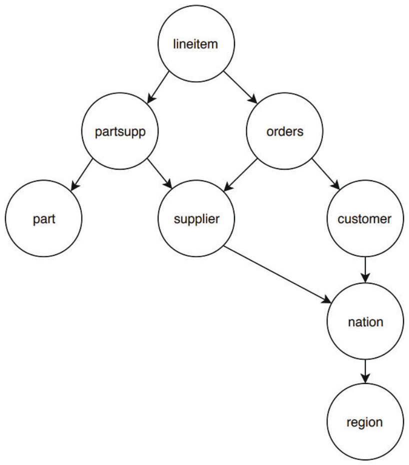


Figure 2: Join Graph for correlated sampling

table holds the most many-to-one relationship it is selected as source relations. In this paper, we only look at source query estimations. So the chosen join graph is

```
Lineitem -> Orders, Orders->Customer, Customer->Nation, Nation->Region
```

. First, We use Simple random sampling without replacement (SRSWOR) on the source table Lineitem table to create the S_1^*. We then join the S_1^* with the Order table to create the second sample S_2. To create the S_3 we join S_2 with the Customer table and we create S_4 by joining S_3 with Nation. In the end, we create S_5 by joining S_4 with the Region table to completely create all the required correlated samples for our experiment.

4 Bootstrap for Join Query Error Estimation

4.1 Methodology

In AQP, a sample of a dataset is taken and queries are run on the sample to get the approximate results faster. There have been research on numerous query selection method [7], but in this paper, we focus on the join selection query listed below:

```
SELECT Aggregation(attribute collection) FROM Table1, Table2 ,...
where join_condition and filter_conditions
```

The Sample S is taken from the dataset and then the query is run on each sample tuple producing query result y_i based on the aggregation functions like COUNT, SUM, AVG, etc. One of the aggregation functions, COUNT is used to get the estimation of the number of rows in the sample data. The regular bootstrapping method resamples all the attributes of the sample data but the optimized bootstrapping method proposed in [2] takes a step before resampling to make the overhead of bootstrap sampling less. Instead of resampling the sample and running the query on each resample, we run the query against the sample S and get the result y_i based on the selection conditions. If the selection condition satisfies y_i will be 1 and if it does not satisfy it will be 0. Then the query result Y_s is calculated by using the following formula:

$$Y_s = \sum_{i=1}^{n} y_i$$

where n is the sample size. We can also estimate the query result ground truth using the following formula:

$$\hat{Y} = \frac{Y_s}{f} \tag{2}$$

where f = $\frac{n}{N}$ is the sample fraction. The query results in Y_s are then saved in an array for bootstrap sampling.

After the sample query result, S_Q is obtained by executing the query Q on the sample relation S, then the bootstrap samples can be generated for error estimates. To calculate an estimation of bootstrap samples we use the following formula:

$$\hat{Y_j} = \frac{Y_j}{f} \tag{3}$$

where Y_j is the query result of the jth bootstrap sample and f is the sample ratio.

4.2 Confidence Interval for Selection AQP

There are different methods to calculate confidence interval [3] such as percentile method, normal approximation method, bias-corrected percentile, etc. In this paper, we look at the standard method of calculating confidence intervals. For every bootstrap sample, we find the desired statistics like mean, and median, and we find the standard deviation which will be the error of estimates.

4.3 Implementation

As proposed in the paper [2] we have a prototype AQP system with the ability to generate error estimations using bootstrap sampling. The system consists of a simple query processor, and a sampler that implements correlated sampling to create join preserved samples. Bootstrap sampler which is used for error estimation. In this paper, we look at the following query formula:

```
SELECT Aggregation(attribute)
FROM table1, table2 ...
WHERE join conditions and selection conditions
```

The simple query processor reads the query structure from a plain text file and based on the join conditions it chooses the sample created using Correlated sampling. Usually after selecting the sample, we would use bootstrap sampling on the sample which will replicate B number of resamples. As all the attributes of the sample are resampled using a simple random sample with replacement (SRSWR), it can cause huge overhead so we utilize the optimized bootstrap sampling method proposed in Cal's paper [2]. In this optimized approach we run the Query Q on the sample S to generate the Sample Query Result S_Q. The query processor will read the query structure and based on the join conditions select the sample and then check each row of the sample to see if the row satisfies the selection condition or not. If the row satisfies the condition then 1 is added to an array and if it does not satisfy the condition 0 is added. This way we will not have to resample whole attributes of the sample and can just resample the array containing 0 and 1. After resampling the array for B a number of times we calculate the statistics of the bootstrap. First, we find the mean of the bootstrap sample. Then we calculate the standard deviation to calculate the Confidence Interval. In this experiment, we use a confidence interval of 95% with the z value of 1.96. Then we calculate the hit ratio to calculate the accuracy of the bootstrap.

5 Experiment and Evaluation

5.1 Experiment Setup

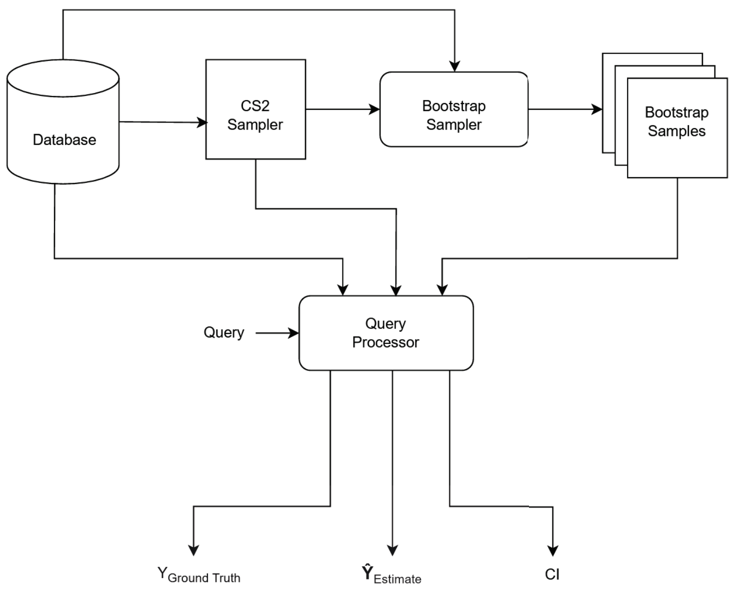


Figure 3: Prototype AQP System

The experiment was performed on the virtual machine with an Intel(R)

Core(TM) i7-10710U CPU which operates at a base frequency of 1.10GHz. The system is equipped with 21GB of RAM which runs on Ubuntu 22.04.2 LTS. The experiment code is written in Rust and Python Language.

The data sets used for experiments are generated using the TPC-H benchmark [8]. We selected different sizes of data sets including 100MB, 1GB, and 10GB. We have chosen four join relationships 3.1. For each join relationship, we have 10 test queries which will be run 10 times each. Testing queries for each join are located in the Appendix part of this paper. We use the sampling fraction of 0.1%, 0.5%, and 1.0%. The experimental program checks if the ground truth that we get from running the query on the original database is between the upper and lower confidence interval. If the ground truth of the original database is between the upper and lower bound then it is a hit otherwise a miss. This was done for each query for each join.

Figure 3 is the prototype AQP system implemented in Rust which consists of four main components: a database, a CS2 sampler, a bootstrap sampler, and a query processor. The CS2 sampler uses the rusqlite crate to interact with the database and create join preserved correlated samples, which is used to calculate the sample estimate. Then the bootstrap sampler selects CS2 samples from the database based on the join conditions and uses the optimized bootstrap method [2] to create bootstrap samples, which are used to calculate the confidence interval and hit ratio. The query processor takes in the query and helps to select CS2 samples based on join conditions and calculate optimized bootstrap samples.

The AQP system works first by parsing the query and creating a plan for execution. Then the CS2 sampler creates correlated samples and returns them to the query processor which forwards the correlated samples to the bootstrap sampler. The bootstrap sampler uses the correlated samples and uses the optimized bootstrap method to create bootstrap samples. The AQP system calculates mean, standard deviation, and confidence interval to estimate the accuracy.

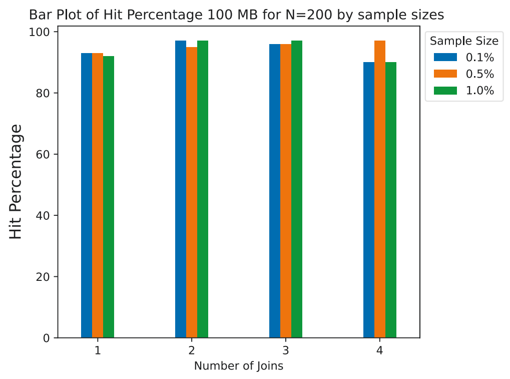
5.2 Accuracy Evaluation

The accuracy evaluation of the experiment is done by calculating the hit ratio. We calculate the hit ratio by the following formula:

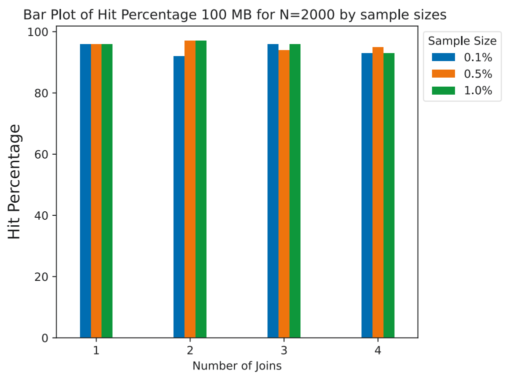
$$\text{hit ratio} = \frac{\text{times(CI hits)}}{\text{times(Total experiments)}} \times 100\% \tag{4}$$

We run the experiment on TPC-H benchmark datasets of three different sizes 100MB, 1GB, and 10 GB. Four different join queries were tested, each with 10 different selection predicates as listed in the Appendix. For each join query, correlated samples were generated at sample sizes 0.1%, 0.5%, and 1.0% of the base data. The query was executed on these samples to obtain the sample query results.

Bootstrap sampling was then performed on the samples to calculate the confidence interval and hit ratio. Two bootstrap samples were tested, 200 and 2000 replicates. Each experiment consisted of running 10 different selection queries for each join query 10 times on a dataset at a specific sample size and bootstrap replicate.

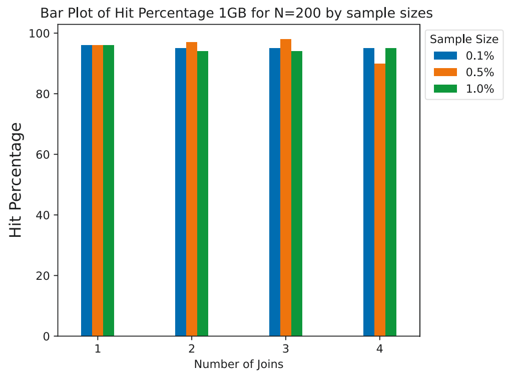


(a) 100MB, B=200

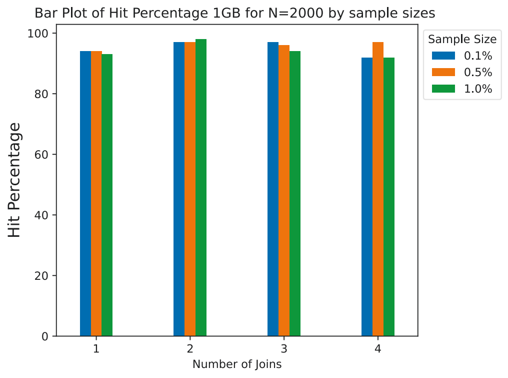


(b) 100MB, B=2000

Figure 4: Hit percentage for 100 MB data set with different bootstrap samples

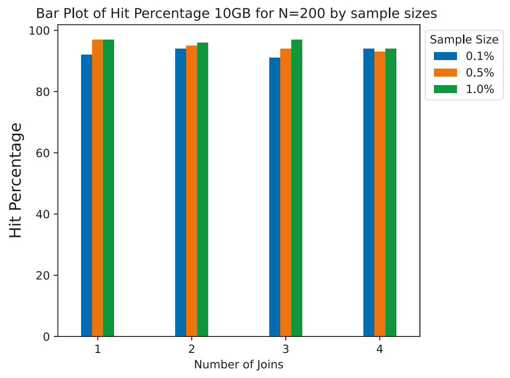


(a) 1GB, B=200

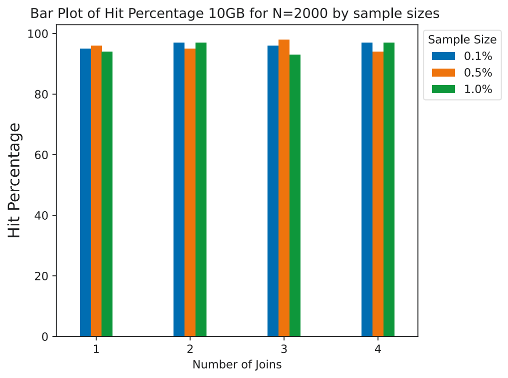


(b) 1GB, B=2000

Figure 5: Hit percentage for 1GB data set with different bootstrap samples



(a) 10GB, B=200



(b) 10GB, B=2000

Figure 6: Hit percentage for 10GB data set with different bootstrap samples

5.2.1 Effect of Bootstrap Sample Size

Figure 4, 5, and 6 compare the hit ratio for 200 vs 2000 bootstrap sample size across the different dataset sizes and sample percentages. The results demonstrate that using a larger number of bootstrap samples improves the accuracy of the estimation. The hit ratio increases from 200 to 2000 bootstrap samples in nearly all cases. This aligns with the expectation, as more bootstrap samples provide a better approximation of the result distribution.

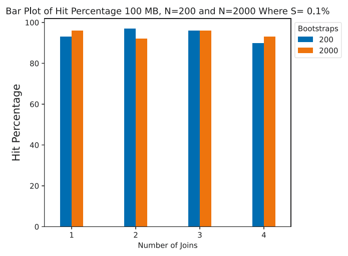
5.2.2 Effect of Sample Size

As seen in figures 7, 8, and 9 the hit ration improves as the sample size increases from 0.1% to 1.0% of the base data. The increase in sample size captures more of the characteristics of the full data leading to more accurate estimates and error bounds.

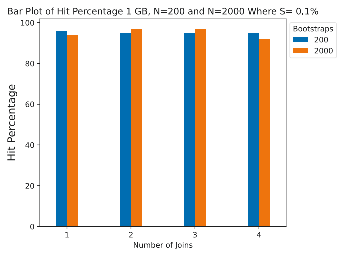
5.2.3 Effect of Dataset Size

Figures 10, 11, and 12 plot the hit ratios for the 100 MB, 1 GB, and 10 GB dataset sizes across the different joins. The results reveal that the accuracy remains reasonably stable even for the small dataset of 100 MB. However, we see slight improvement for the larger dataset of 10 GB. So while larger datasets do improve accuracy, the bootstrap approach maintains robust performance even for the smallest datasets.

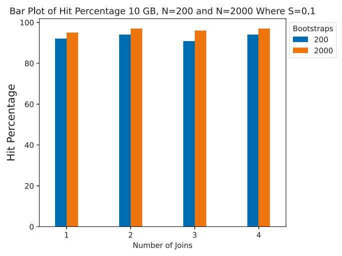
Overall, The hit ratio consistently stays above 90% validating that using



(a) 100 MB S=0.1%

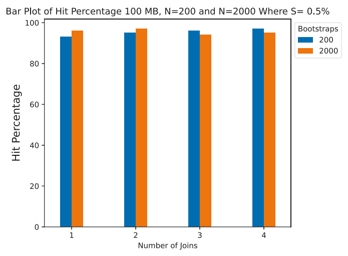


(b) 1 GB S=0.1%

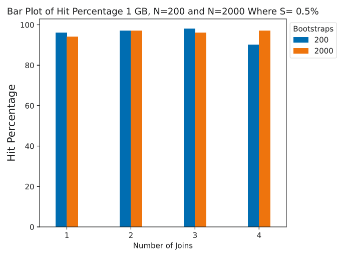


(c) 10GB S=0.1%

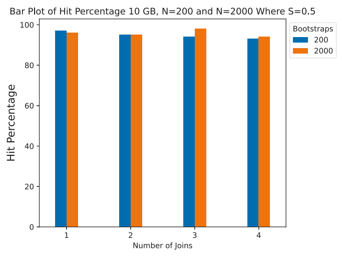
Figure 7: Hit percentage for 0.1% sample size and different data sizes



(a) 100 MB S=0.5%

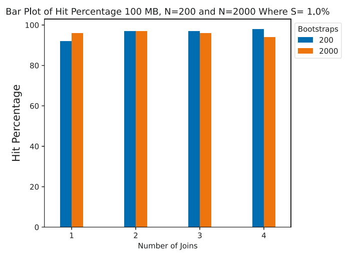


(b) 1 GB S=0.5%

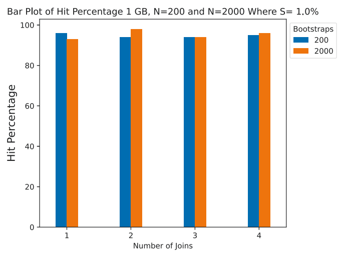


(c) 10GB S=0.5%

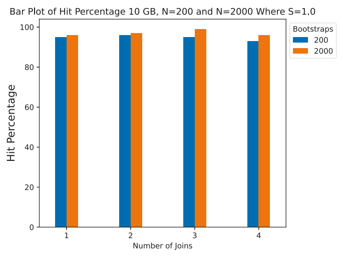
Figure 8: Hit percentage for 0.5% sample size and different data sizes



(a) 100 MB S=1.0%



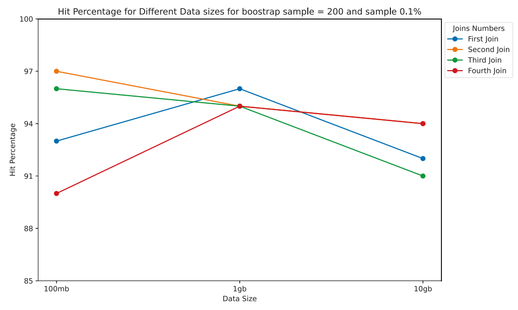
(b) 1 GB S=1.0%



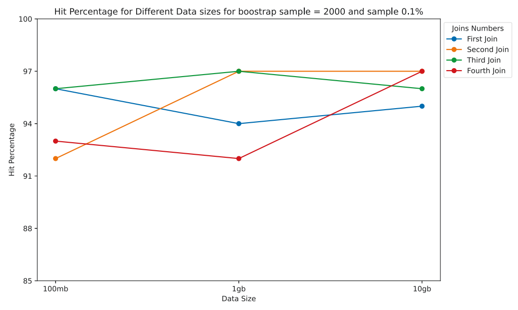
(c) 10GB S=1.0%

Figure 9: Hit percentage for 1% sample size and different data sizes

bootstrap sampling with correlated sampling can successfully estimate errors for join queries in the AQP system. The accuracy evaluation shows that correlated bootstrap sampling works well across varied sample sizes, dataset sizes, and bootstrap numbers.

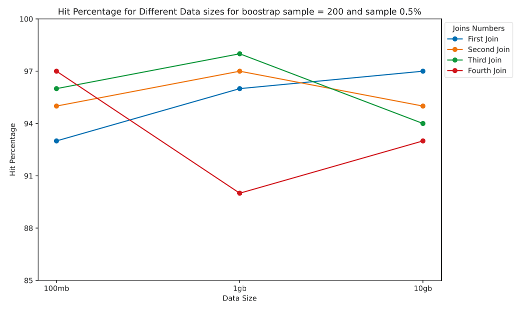


(a) B=200 and S=0.1%

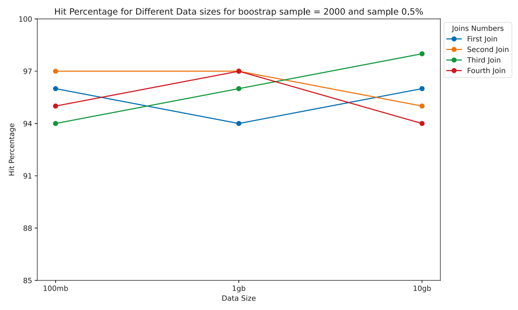


(b) B=2000 and S=0.1%

Figure 10: Line Graph for Different data sizes for sample size =0.1

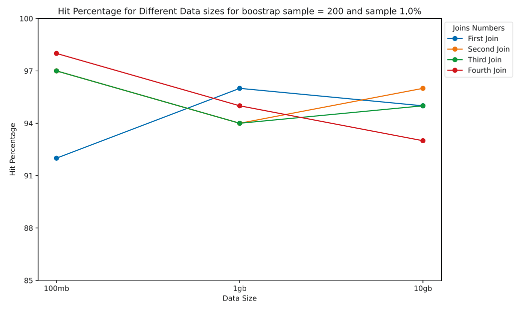


(a) B=200 and S=0.5%

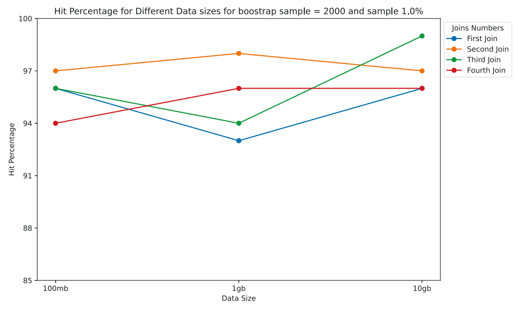


(b) B=2000 and S=0.5%

Figure 11: Line Graph for Different data sizes for sample size =0.5



(a) B=200 and S=1.0%



(b) B=2000 and S=1.0%

Figure 12: Line Graph for Different data sizes for sample size =1.0

6 Conclusion

The experiment evaluated the accuracy of approximate query processing (AQP) systems for join queries using correlated sampling with bootstrap techniques. The results on TPC-H benchmark data confirm the efficacy of this approach for producing join-preserving samples and accurate error estimates.

The graphs highlight how the bootstrap hit percentage, measuring how often the true statistic falls within the bootstrap confidence interval, improves with larger sample sizes, data sets, and number of replicates. However, accuracy remains reasonably high even for smaller sample sizes and datasets, with hit percentages consistently above 90%.

Overall, the high hit percentages demonstrate the robust performance of the bootstrap approach across varied conditions. The graphs validate that correlated sampling coupled with bootstrap resampling can successfully estimate join query result distributions and error bounds. These techniques show promise for enabling accurate approximate query processing on complex joins. Future research can be based on evaluating this methodology for more complicated join queries with multiple selection conditions.

References

[1] Barrett, S. How much data is produced every day in 2022. *TECHTREND* (April 2021).

[2] Cal, S., Cheng, E., and Yu, F. Optimized bootstrap sampling for σ-aqp error estimation: A pilot study. In *SEDE* (2021).

[3] Campbell, M. K., and Torgerson, D. J. Bootstrapping: estimating confidence intervals for cost-effectiveness ratios. *QJM 92*, 3 (1999), 177–82.

[4] Chernick, M. R., and LaBudde, R. A. *An introduction to bootstrap methods with applications to R.* John Wiley & Sons, 2014.

[5] Efron, B. Bootstrap methods: another look at the jackknife. *The Annals of Statistics* (1979), 1–26.

[6] Efron, B., and Tibshirani, R. J. *An Introduction to the Bootstrap.* CRC Press, 1994.

[7] Markovic, J., and Taylor, J. Bootstrap inference after using multiple queries for model selection. *arXiv preprint arXiv:1612.07811* (2017).

[8] (TPC), T. P. P. C. Tpc benchmark™ h (decision support) standard specification. Tech. rep., Transaction Processing Performance Council, Presidio of San Francisco Building 572B Ruger St. (surface) P.O. Box 29920 (mail) San Francisco, CA 94129-0920, 2014. Revision 2.17.1.

[9] Wilson D., S. Hou, W. C., and F., Y. Scalable correlated sampling for join query estimations on big data. *SEDE 64* (2019), 41–50.

Appendix A Test Queries for First Join

```
SELECT count(*) FROM lineitem, orders WHERE l_orderkey = o_orderkey

AND L_LINENUMBER < 3 AND L_LINENUMBER > 0;


SELECT count(*) FROM lineitem, orders WHERE l_orderkey = o_orderkey

AND L_LINENUMBER < 5 AND L_LINENUMBER > 2;


SELECT count(*) FROM lineitem, orders WHERE l_orderkey = o_orderkey

AND L_DISCOUNT < .07 AND L_DISCOUNT > .02;


SELECT count(*) FROM lineitem, orders WHERE l_orderkey = o_orderkey

AND L_EXTENDEDPRICE < 100000.00 AND L_EXTENDEDPRICE > 20000.00;


SELECT count(*) FROM lineitem, orders WHERE l_orderkey = o_orderkey

AND L_DISCOUNT < .04 AND L_DISCOUNT > 0.0;


SELECT count(*) FROM lineitem, orders WHERE l_orderkey = o_orderkey

AND L_QUANTITY < 20 AND L_QUANTITY > 10;


SELECT count(*) FROM lineitem, orders WHERE l_orderkey = o_orderkey

AND L_DISCOUNT < .05 AND L_DISCOUNT > .02;
```

```
SELECT count(*) FROM lineitem, orders WHERE l_orderkey = o_orderkey

AND L_EXTENDEDPRICE < 15000.00 AND L_EXTENDEDPRICE > 0.0;


SELECT count(*) FROM lineitem, orders WHERE l_orderkey = o_orderkey

AND o_totalprice < 120000 AND o_totalprice > 60000;


SELECT count(*) FROM lineitem, orders WHERE l_orderkey = o_orderkey

AND o_totalprice < 180000 AND o_totalprice > 100000;
```

Appendix B Test Queries for Second Join

```
SELECT count(*) FROM lineitem, orders, customer WHERE l_orderkey = o_orderkey
AND o_custkey = c_custkey AND L_LINENUMBER < 3 AND L_LINENUMBER > 0;

SELECT count(*) FROM lineitem, orders, customer WHERE l_orderkey = o_orderkey
AND o_custkey = c_custkey AND L_DISCOUNT < .07 AND L_DISCOUNT > .02;

SELECT count(*) FROM lineitem, orders, customer WHERE l_orderkey = o_orderkey
AND o_custkey = c_custkey AND L_EXTENDEDPRICE < 100000.00 AND
L_EXTENDEDPRICE > 20000.00;

SELECT count(*) FROM lineitem, orders, customer WHERE l_orderkey = o_orderkey
AND o_custkey = c_custkey AND L_DISCOUNT < .04 AND L_DISCOUNT> 0.0;

SELECT count(*) FROM lineitem, orders, customer WHERE l_orderkey = o_orderkey
AND o_custkey = c_custkey AND L_QUANTITY < 20 AND L_QUANTITY > 10;

SELECT count(*) FROM lineitem, orders, customer WHERE l_orderkey = o_orderkey
AND o_custkey = c_custkey AND L_EXTENDEDPRICE < 15000.00 AND
L_EXTENDEDPRICE > 0.0;
```

```
SELECT count(*) FROM lineitem, orders, customer WHERE l_orderkey = o_orderkey

AND o_custkey = c_custkey AND o_totalprice < 120000 AND o_totalprice > 60000;


SELECT count(*) FROM lineitem, orders, customer WHERE l_orderkey = o_orderkey

AND o_custkey = c_custkey AND o_totalprice < 180000 AND o_totalprice > 100000;


SELECT count(*) FROM lineitem, orders, customer WHERE l_orderkey = o_orderkey

AND o_custkey = c_custkey AND c_acctbal < 5000 AND c_acctbal > 2000;


SELECT count(*) FROM lineitem, orders, customer WHERE l_orderkey = o_orderkey

AND o_custkey = c_custkey AND c_acctbal < 10000 AND c_acctbal > 5000;
```

Appendix C Test Queries for Third Join

```
SELECT count(*) FROM lineitem, orders, customer, nation

WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey

AND c_nationkey = n_nationkey AND L_LINENUMBER < 3 AND L_LINENUMBER > 0;


SELECT count(*) FROM lineitem, orders, customer, nation

WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey

AND c_nationkey = n_nationkey AND L_DISCOUNT < .07 AND L_DISCOUNT > .02;


SELECT count(*) FROM lineitem, orders, customer, nation

WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey

AND c_nationkey = n_nationkey AND L_EXTENDEDPRICE < 100000.00

AND L_EXTENDEDPRICE > 20000.00;


SELECT count(*) FROM lineitem, orders, customer, nation

WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey

AND c_nationkey = n_nationkey AND L_DISCOUNT < .04

AND L_DISCOUNT> 0.0;


SELECT count(*) FROM lineitem, orders, customer, nation

WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey

AND c_nationkey = n_nationkey AND L_QUANTITY < 20
AND L_QUANTITY > 10;

SELECT count(*) FROM lineitem, orders, customer, nation
WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey
AND c_nationkey = n_nationkey AND L_EXTENDEDPRICE < 15000.00
AND L_EXTENDEDPRICE > 0.0;

SELECT count(*) FROM lineitem, orders, customer, nation
WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey
AND c_nationkey = n_nationkey AND o_totalprice < 120000
AND o_totalprice > 60000;

SELECT count(*) FROM lineitem, orders, customer, nation
WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey
AND c_nationkey = n_nationkey AND o_totalprice < 180000
AND o_totalprice > 100000;

SELECT count(*) FROM lineitem, orders, customer, nation
WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey
AND c_nationkey = n_nationkey AND c_acctbal < 5000
AND c_acctbal > 2000;

SELECT count(*) FROM lineitem, orders, customer, nation
```

```
WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey

AND c_nationkey = n_nationkey AND c_acctbal < 10000

AND c_acctbal > 5000;
```

Appendix D Test Queries for Fourth Join

```
SELECT count(*) FROM lineitem, orders, customer, nation, region

WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey

AND c_nationkey = n_nationkey AND n_regionkey = r_regionkey

AND L_LINENUMBER < 3 AND L_LINENUMBER > 0;


SELECT count(*) FROM lineitem, orders, customer, nation, region

WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey

AND c_nationkey = n_nationkey AND n_regionkey = r_regionkey

AND L_DISCOUNT < .07 AND L_DISCOUNT > .02;


SELECT count(*) FROM lineitem, orders, customer, nation, region

WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey

AND c_nationkey = n_nationkey AND n_regionkey = r_regionkey

AND L_EXTENDEDPRICE < 100000.00 AND L_EXTENDEDPRICE > 20000.00;


SELECT count(*) FROM lineitem, orders, customer, nation, region

WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey

AND c_nationkey = n_nationkey AND n_regionkey = r_regionkey

AND L_DISCOUNT < .04 AND L_DISCOUNT> 0.0;
```

```
SELECT count(*) FROM lineitem, orders, customer, nation, region

WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey

AND c_nationkey = n_nationkey AND n_regionkey = r_regionkey

AND L_QUANTITY < 20 AND L_QUANTITY > 10;


SELECT count(*) FROM lineitem, orders, customer, nation, region

WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey

AND c_nationkey = n_nationkey AND n_regionkey = r_regionkey

AND L_EXTENDEDPRICE < 15000.00 AND L_EXTENDEDPRICE > 0.0;


SELECT count(*) FROM lineitem, orders, customer, nation, region

WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey

AND c_nationkey = n_nationkey AND n_regionkey = r_regionkey

AND o_totalprice < 120000 AND o_totalprice > 60000;


SELECT count(*) FROM lineitem, orders, customer, nation, region

WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey

AND c_nationkey = n_nationkey AND n_regionkey = r_regionkey

AND o_totalprice < 180000 AND o_totalprice > 100000;


SELECT count(*) FROM lineitem, orders, customer, nation, region

WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey

AND c_nationkey = n_nationkey AND n_regionkey = r_regionkey
AND c_acctbal < 5000 AND c_acctbal > 2000;


SELECT count(*) FROM lineitem, orders, customer, nation, region

WHERE l_orderkey = o_orderkey AND o_custkey = c_custkey

AND c_nationkey = n_nationkey AND n_regionkey = r_regionkey

AND c_acctbal < 10000 AND c_acctbal > 5000;
```