Correlated Sample Synopsis on Big Data

by

David S. Wilson

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

requirements for the degree of

Master of Science

in the

Computer Information Systems

Program

YOUNGSTOWN STATE UNIVERSITY

December, 2018

Correlated Sample Synopsis on Big Data

David S. Wilson

Signature:

David Wilson, Student	Date

Approvals:

Dr. Feng Yu, Thesis Advisor	Date
Dr. Yong Zhang, Committee Member	Date
Dr. John R Sullins, Committee Member	Date
Dr. Salvatore A. Sanders, Dean of Graduate Studies	Date

Abstract

Correlated Sample Synopsis (or CS2) has been proven to be a valuable option concerning centralized databases but has yet to be tested on big data. With the overall accumulation of data growing at an alarming rate, scalable query estimation and approximate query processing are becoming necessary for large databases. Query estimations based on the Simple Random Sample Without Replacement (or SRSWOR) return results with extremely high relative errors for join queries. Existing methods, such as Join Synopses, only work well with foreign key joins, and the sample size can grow dramatically as the dataset gets larger. This research aims to provide that CS2 can speed up search query length results, give precise join query estimations, and minimize storage costs when presented with big data. In addition, this research extends the correlated sampling techniques and estimation methods of CS2 to the big data environment with no index present. Extensive experiments with large TPC-H datasets in Apache Hive show that CS2 produces fast and accurate query estimations on big data.

Acknowledgments

First and foremost, I would like to direct my uppermost gratitude to my advisor, Dr. Feng Yu. I appreciate the opportunity he provided me with the ability to work with him, as well as the support, patience, and knowledge he supplied to help me complete my research.

I also would like to thank my committee members Dr. John Sullins, and Dr. Yong Zhang for taking time out of their schedule to share their insights and knowledge. I would also like to give a special thanks to Dr. Alina Lazar and Ms. Connie Frisby, who both were instrumental during my undergrad and graduate studies. I would like to thank the Cushwa Shearing Fellowship for giving me the opportunity to do research in my field, as well as giving me the financial support.

I would also like to thank my family and friends for giving me a solid foundation to grow in my studies, and as a person.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation and Overview

Data is everywhere. We produce 2.5 quintillion bytes (2.5 billion Gigabytes) of data each day. Ninety percent of the data created in the world has been created in the past two years [3]. To put that into perspective, IBM created the IBM Model 350 Disk File in 1956. It was the size of a compact-size car, and had a storage capacity of five megabytes. If one were to place these machines side by side, based off the amount of data we use in one day, they would circle the earth nine thousand one hundred and ninety times. With the sizes of company databases reaching terabytes and even petabytes, and at the speed of which this data is being accumulated, the need for query optimization has never been so high.

Query optimization [4] is the process of using statistics about the database, as well as assumptions about the attribute values, to acquire the best execution plans for queries. Some databases are large, and data streams in so fast that queries can take minutes, hours, even days to process. CS2 (Correlated Sample Synopsis) [2] is a statistical summary for a database, and through unique methods, aims to provide a fast and precise result size estimation for queries with joins and arbitrary selections. The aim of this thesis is to provide background information on Big Data, CS2, SRSWOR (Simple Random Sample Without Replacement), Apache Hadoop and Apache Hive, and Join Synopsis, as well as taking the methods of CS2 and apply them to big data sets and presenting the findings.

1.2 Organization

The thesis is organized as follows. Starting with Section 2.1, Section 2 explains background information on Big Data, discussing the origins, as well as the Three V's and additional V's. In Section 2.2 Map-Reduce, HDFS, YARN, and Hadoop Common modules of Apache Hadoop and Hive are introduced. Section 2.3 covers the structure and use of a Join Graph of a Database. Section 2.4 deals with SRSWOR (Simple Random Sample Without Replacement), and the problems that occur when attempting to use SRSWOR individually on relations. Join Synopses methods are explained in Section 2.5, with its certain complications due to its foundation being discussed. Section 3 briefs about CS2, with the construction of CS2 being explained in Section 3.1 and the type of join queries it uses in Section 3.2. In Section 4 the experiments and results are introduced. The setup is presented (hardware and software) in Section 4.1. The step-by-step process of creating the datasets, as well as preparation of the datasets for experimentation is discussed in Section 4.2. Section 4.3 holds the results from the experiments and explains the differences experienced from using two datasets. Additionally all code used in experimentation is located in Appendix A and Appendix B. Section 5 concludes the thesis and discusses future works.

2 Background

2.1 Big Data

Big Data may just be one of the most misunderstood terms in the technology field. It is miscommonly referred to as a large volume of data. While not entirely incorrect, there is much more to Big Data then just size. In the following sections, the origins of big data, as well as what defines data as “Big data”, will be discussed.

2.1.1 Brief History of Big Data

The term “Big Data” was first coined in 1998 by John Mashey of Silicon Graphics, Inc., although this is debated [5]. Others had written about big data before this date, but Mr.Mashey was the first that used the term in the context of computing. Even though the term Big Data was created in the 90’s it was not until the early 2000’s that it took the form of what is is considered today. In February 2001, Doug Laney created the three V’s of Big Data, which are Volume, Variety, and Velocity. [6]

2.1.2 The Three V’s

The first V, Volume, is what most people think of when they discuss big data. Some may ask, what size is considered “Big Data”? There is no automatic answer to this question, as the size of the storage of data changes over time. In the 90’s and 2000’s big data would have been considered gigabytes, where as today big data could be considered terabytes, or even petabytes. And sometime in the future terabytes and petabytes most likely will not be considered big data, as exabytes and zettabytes will take their place. [7]

Variety, the second V, involves types of data. Not all data is created equal. Data can be considered structured, such as data in a relational database. In this case the information is highly organized, in which it is easy to find and analyze the data. Some examples of unstructured data would be audio/video files, word documents, and emails. This type of data is hard to organize, and while some forms of unstructured data may have a an internal architecture to them, being able to correlate with other data may prove to be difficult.

The third V, Velocity, handles not only the speed in which data accumulates, but the speed in which it would take to analyze the data. Figure 1 shows transactions per year for two major retail chains, Home Depot and Lowe's. Lowe's in 2017 had 953 million transactions during the year, or an average of 1813 transactions a minute. Home Depot did even more business with 1.579 billion transactions a year, averaging out at 3004 transactions a minute. These are enormous amounts of data streaming into the system at one time. Attempting to analyze this amount data to research customer trends, as well as other business analytics, would take large processing power.

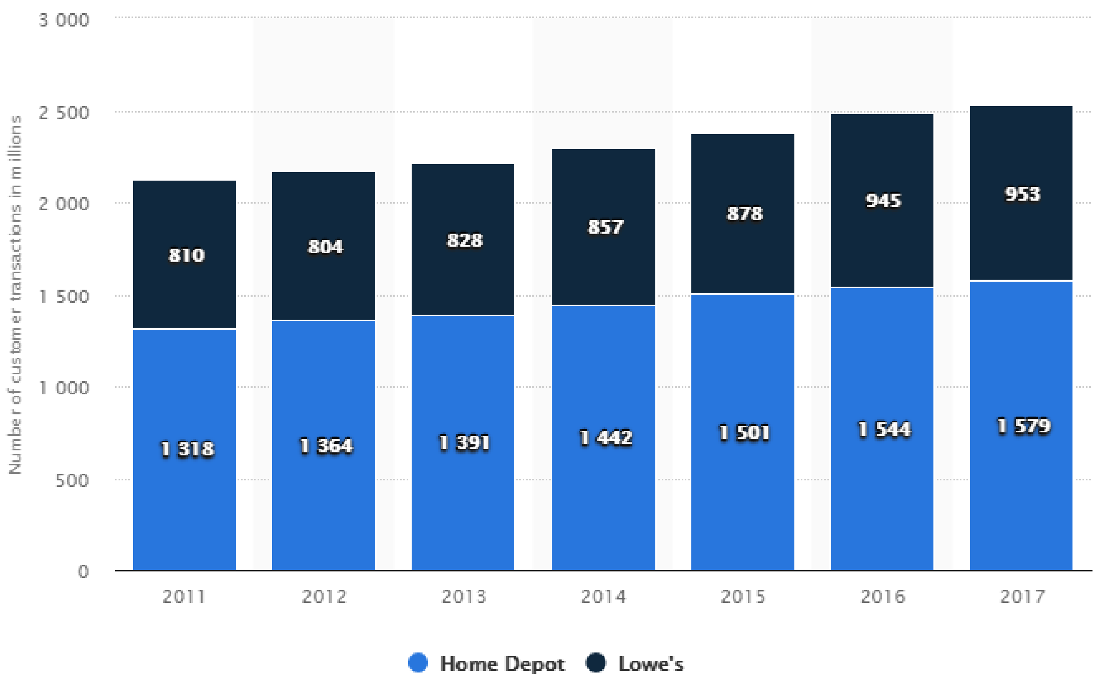


Figure 1: Transactions Per Year [1]

2.1.3 Additional V's

In the past five to ten years, two additional V's have supplemented the original three V's. The creator and exact creation date is not known. Some establishments recognize just the three, while others four, and even others all five V's.

The fourth V, Veracity, deals with the accuracy and certainty of your data. According to IBM [8], one in three business leaders do not trust the data given when using to make a decision. With poor data quality costing companies worldwide over 3.1 trillion dollars a year, one could understand why. When dealing with big data, accuracy of data becomes instrumental.

Value, the fifth V, considers the usefulness of the data. Is the return on data worth the cost of storing? Storing pentabytes of data, when nothing is usable for extraction, is not cost efficient for a business. [9]

2.2 Apache Hadoop and Hive

Created in 2005 by Doug Cutting and Mike Cafarella, Apache Hadoop is an open source distributed processing framework that manages data processing as well as provides storage for big data applications running in clustered systems.

Apache Hive is an open source tool built by developers at Facebook in 2007. Hive, running on the Apache Hadoop framework, allows developers to use HQL (Hive Query Language) on Hadoop's HDFS and Map-Reduce [10].

2.2.1 Hadoop

The Hadoop framework [11] consists of multiple modules, each having its own distinctive responsibilities.

Hadoop Common is the storehouse for other Hadoop Modules. It holds all of the files in which the other Hadoop Modules need to run properly.

Hadoop Distributed File System, or HDFS for short,[12] deals with the storage of data of an Hadoop Cluster. A major issue with storing streaming, large sets of data, is hardware failure. HDFS is built to combat this, by using a process called replication. HDFS consists of a name node, which stores all meta data of all

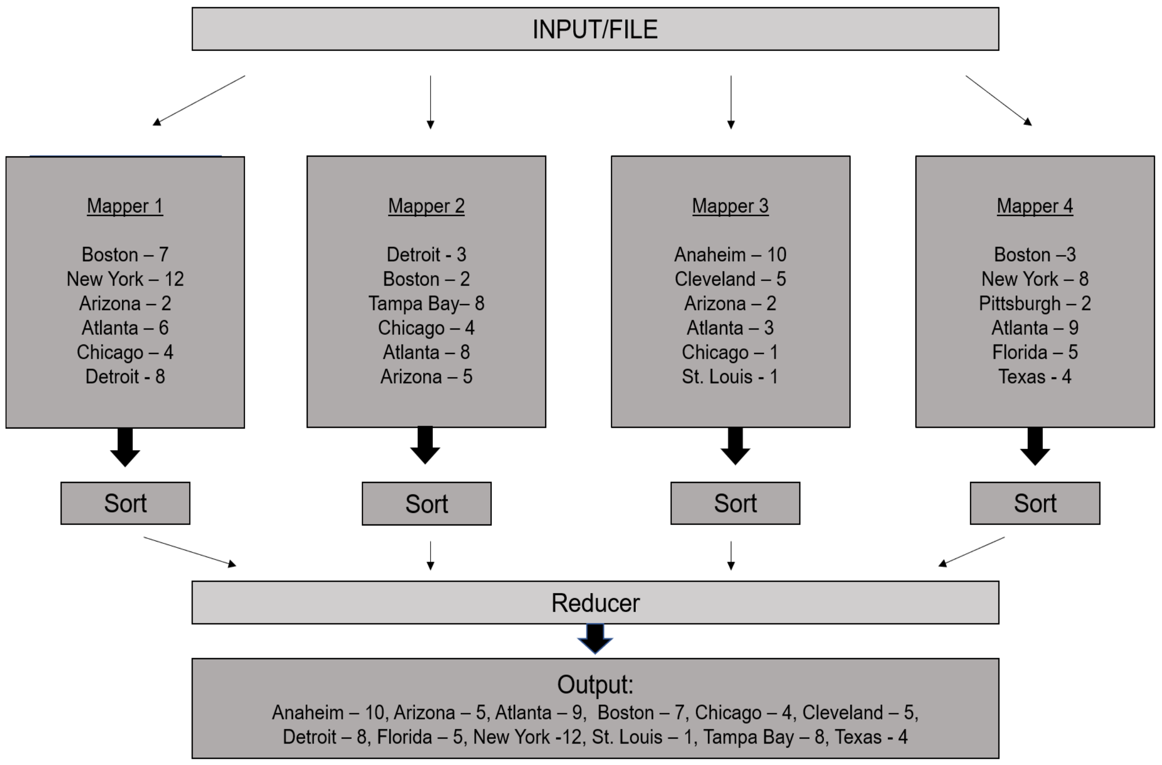


Figure 2: An Example of Map-Reduce

files stored. It also has data nodes as well, which hold all of the actual data. Each data node consists of a multitude of blocks, with each block of data being stored into 3 different data node locations in the cluster. If at any time there is a node that fails, or a machine in the cluster fails, another block copy is made on another node or machine [10].

Hadoop YARN (Yet Another Resource Negotiator) [13], manages resources of other Hadoop Modules, as well as schedules tasks and sets nodes for other Hadoop Modules to execute on [14].

There are two tasks/functions for Hadoop Map-Reduce. The Map function takes input data and creates key/value pairs from it. It then shuffles and sorts the data. The Reduce Function then takes the key/value pairs from all of the mappers and combines them before reducing the key/value pairs [15].

Figure 2 shows an example of the Map-Reduce process. A user has a large file, consisting of two columns. The first column holds a sports team city (key), and the second column holds how many hits that team hit on a certain day(value). The user would like to know what is the most hits in a day for each sports team. They assign a mapper task in which four mappers then create key/value pairs out of the file. Each mapper then sorts its key/value pairs and sends it to the reducer. The reducer then combines them and reduces the key/value pairs and sends it to the output [15].

2.2.2 Hive

Apache Hive [16] was created to make it easier for users to be able to use Hadoop's Map-Reduce and HDFS without an advanced knowledge of Java.

As mentioned earlier, Hive uses a similar language to SQL, called HQL or Hive Query Language. With the use of this language users are able to perform data queries, as well as summarize and analyze data. Users can use traditional command line to work in Hive, or us HWI (Hive Web Interface). HWI is a graphical user interface, or GUI, that simplifies the use of hive.

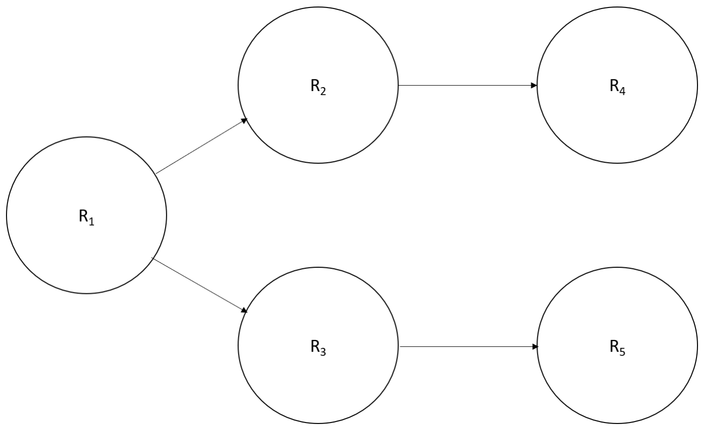


Figure 3: A Basic Join Graph

2.3 Join Graph of a Database

Definition 1. (Join Graph) A join graph [17] is a visual representation of a database in which the flow of joins is explained. It can be created to take into consideration the relational type of joins,(many-to-many, many-to-one, one-to-one), and also if there are multiple attributes that can be used within the join. It is a general representation in which the join relations of a database are mapped out [2].

Definition 2. (Joinable Relations) Two relations considered joinable, R_i and R_k, $i \neq k$, when there is a path ≥ 1 between the relations R_i and R_k [2].

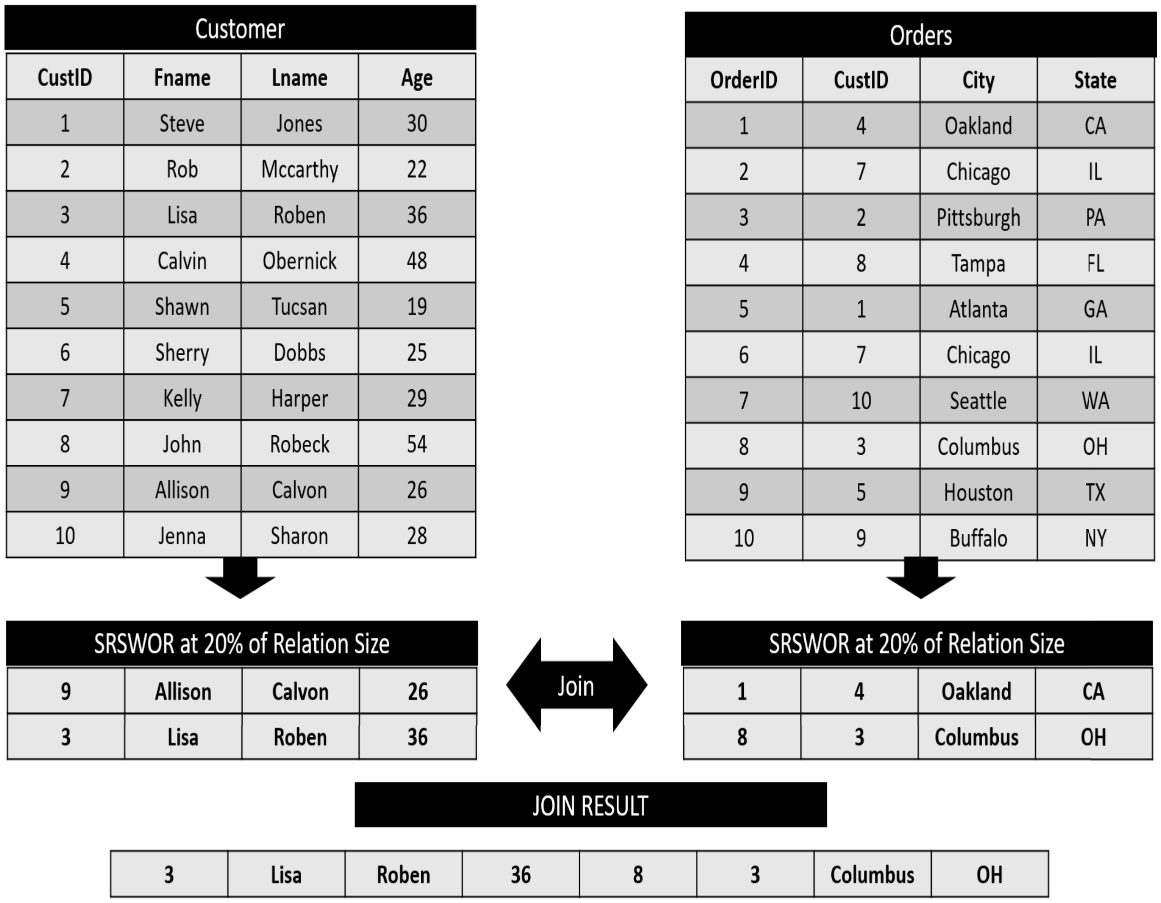
Definition 3. (Joinable Tuples) Under the assumption that R_i and R_k is a joinable relation, a tuple in R_i,denoted by t_i, and a tuple in R_k, denoted t_k, is considered joinable if t_i can find a match t_{i+1} in R_{i+1}, t_{i+1} can find a match t_{i+2} in R_{i+2}, and t_{k-1} can find a match t_k in R_k [2].

Figure 3 is a basic join graph of a database. It shows that Relation 1 denoted as R_1, has joinable attributes with Relation 2, as well as Relation 3, denoted with R_2 and R_3 respectively. R_2 has joinable attributes with Relation 4, denoted with R_4, but does not have any joinable attributes with R_3 or Relation 5, denoted with R_5. R_3 has joinable attributes with R_5, but no joinable attributes with R_2 or R_4.

2.4 Simple Random Sample Without Replacement

Simple Random Sample Without Replacement, or SRSWOR, [18, 19] has previously been tested as a sample synopsis. A SRSWOR of each relation is taken separately, and then the resulting independent samples are joined. Unfortunately, the final results end in massive errors of the join size estimation [20]. SRSWOR is beneficial if one is only seeking to get a size estimation on an individual relation.

An example of independent SRSWOR on two separate joinable relations is shown in Figure 4. This example consists of a Customer Table and Orders table. Creating a SRSWOR at twenty percent the size of the relations results in two random tuples from each table. A join based off the foreign key CustID on the original relations would result in ten tuples. Joining the samples based on the same constraints would result in one tuple. To get the join size estimation one needs to take the number of tuples in the sample and divide this number by the percent of the SRSWOR, which in this case would be twenty percent. The resulting estimation would be five, which is far off from the actual ten in the original relations. Relative error, which is used as a measure of precision, is the ratio of the absolute error of a measurement, to the



Customer

CustID	Fname	Lname	Age
1	Steve	Jones	30
2	Rob	Mccarthy	22
3	Lisa	Roben	36
4	Calvin	Obernick	48
5	Shawn	Tucsan	19
6	Sherry	Dobbs	25
7	Kelly	Harper	29
8	John	Robeck	54
9	Allison	Calvon	26
10	Jenna	Sharon	28

Orders

OrderID	CustID	City	State
1	4	Oakland	CA
2	7	Chicago	IL
3	2	Pittsburgh	PA
4	8	Tampa	FL
5	1	Atlanta	GA
6	7	Chicago	IL
7	10	Seattle	WA
8	3	Columbus	OH
9	5	Houston	TX
10	9	Buffalo	NY

SRSWOR at 20% of Relation Size

9	**Allison**	**Calvon**	**26**
3	**Lisa**	**Roben**	**36**

SRSWOR at 20% of Relation Size

1	**4**	**Oakland**	**CA**
8	**3**	**Columbus**	**OH**

JOIN RESULT

3	**Lisa**	**Roben**	**36**	**8**	**3**	**Columbus**	**OH**

Figure 4: SRSWOR of Two Relations

measurement being taken. In the case of this example the relative error resulted with an abysmal 50. In an attempt to put this into perspective, shown in future results of the thesis experiments, the average relative error was .635.

2.5 Join Synopses

While Join Synopses [21] use SRSWOR in its mechanics, the process does adds a join correlation between individual relations, causing a much better relative error. Join Synopses uses foreign key joins and computes samples of a small set of joins,

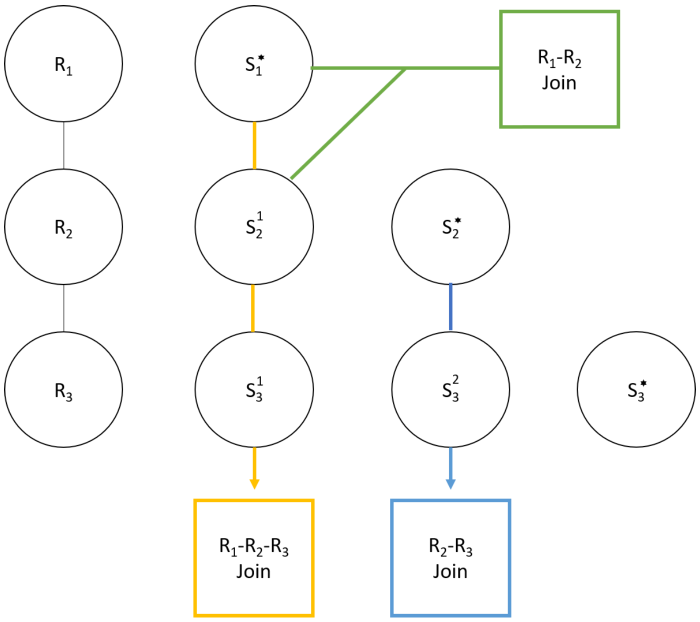


Figure 5: Join Synopses Process

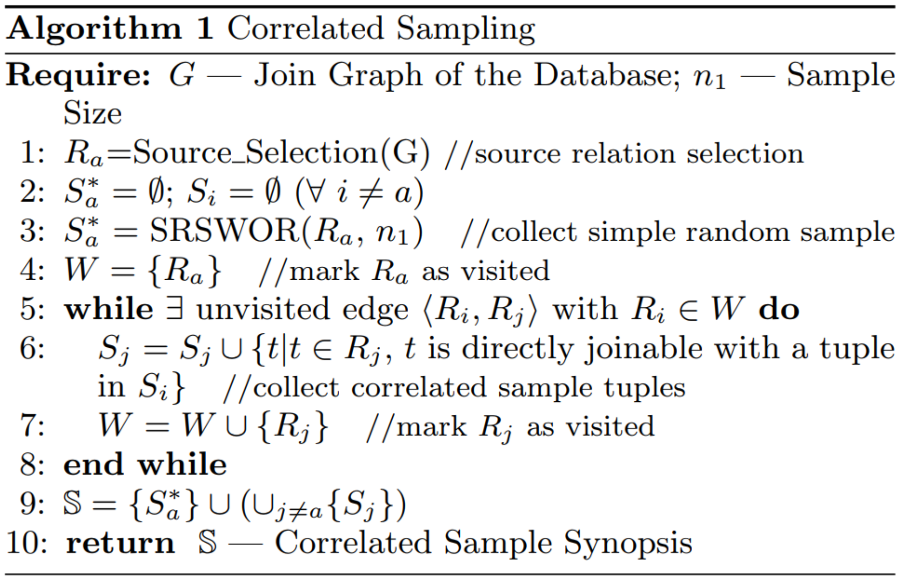
procuring samples of all possible joins in a schema. These samples are then stored, and joined with individual SRSWOR relations to form a unbiased finalized set of correlated random tuples that can be used for query estimation.

Figure 5 shows an example of the Join Synopses process and how preparations for a join estimation is made. For any of the three relations, exactly the same as SRSWOR, a individual relation can be estimated by using just a SRSWOR. These are denoted in the figure as S_1^*, S_2^*, and S_3^*. The relations needed to calculate a join query size estimation of R_1 and R_2, would be a correlated sample of R_1 and R_2, denoted S_2^1, and S_1^*. To get the join query size estimation of a join between R_2 and R_3, a correlated sample of R_2 and R_3, denoted as S_3^2, and S_2^* is needed. Finally, the relations needed to estimate a join query size estimation for R_1, R_2, and R_3, would include S_1^*, S_2^1, and a sample of a correlated join between R_1 and R_3, denoted S_3^1. The issue with Join synopses, which is not present in CS2, is storage expansion. The accumulation of samples grow exponentially as the database grows. Join Synopsis also cannot do many-to-many relations due to its foreign-key structure.

3 CS2 on Big Data

3.1 Construction of CS2

CS2 (or Correlated Sample Synopses) [2] is a statistical summary for a database, which can be used for both query estimation and approximate query processing. The purpose of CS2 is to create a unbiased, fast, and precise estimation for queries with all types of joins and selections. Preserving join relationships between tuples and their relations is the key to CS2.



```
Algorithm 1 Correlated Sampling
Require: $G$ — Join Graph of the Database; $n_1$ — Sample Size
 1: $R_a$=Source_Selection(G) //source relation selection
 2: $S_a^* = \emptyset$; $S_i = \emptyset$ $(\forall\ i \neq a)$
 3: $S_a^* =$ SRSWOR$(R_a, n_1)$  //collect simple random sample
 4: $W = \{R_a\}$  //mark $R_a$ as visited
 5: while $\exists$ unvisited edge $\langle R_i, R_j \rangle$ with $R_i \in W$ do
 6:     $S_j = S_j \cup \{t | t \in R_j$, $t$ is directly joinable with a tuple in $S_i\}$  //collect correlated sample tuples
 7:     $W = W \cup \{R_j\}$  //mark $R_j$ as visited
 8: end while
 9: $\mathbb{S} = \{S_a^*\} \cup (\cup_{j \neq a}\{S_j\})$
10: return $\mathbb{S}$ — Correlated Sample Synopsis
```

Figure 6: Correlated Sampling Algorithm [2]

3.1.1 Join Graph Path Creation and Source Relation Selection

Correlated Sampling begins with the creation of a join graph of the database, as well as the determined size preferred for the sample relations. A source relation selection must then be made.

It is important to note, CS2 does work with any join relationship (one-to-many, many-to-one, many-to-many). However, when selecting your sampling path and source relation it is suggested to use and follow a many-to-one relationship, as using a one-to-many or many-to-many relationship can cause the synopsis to grow considerably, subtracting from the overall number of sample tuples that can be taken from the source relation. For a complicated join graph, multiple source relations are allowed to follow many-to-one-relationships.

3.1.2 Correlated Sample Synopsis

Figure 7 shows a sample example of the process that transpires once the source relation and path selection are decided on. A simple random sample without replacement is performed on the source relation denoted as R_1, with results of this SRSWOR being placed in a sample relation, denoted as S_1^* with a star to signify it is a SRSWOR of R_1. The next relation, denoted R_2 is now ready to be moved to. To create the correlation between relations and preserve the join relationships, S_1^* is joined with R_2, with the results being placed into a second sample relation, denoted as S_2. In this example, the source relation only consists of one edge to another relation. In the

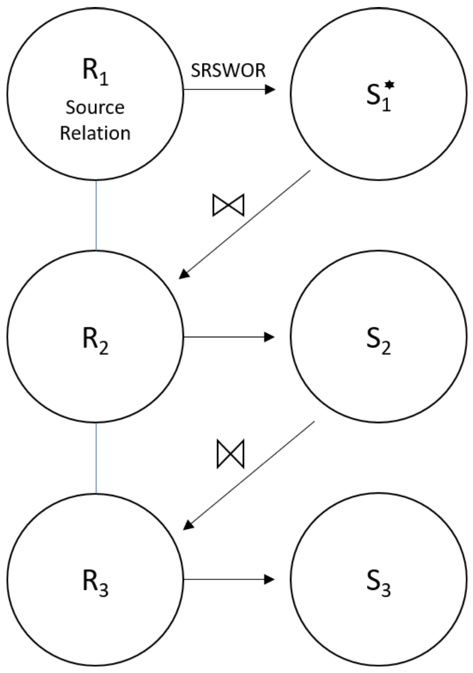


Figure 7: The Process of Creating Correlated Samples Using CS2

case that there are multiple edges to multiple relations, one would exhaust all possibilities by creating sample relations for each relation until all edges are accounted for. Relation three, denoted as R_3, is then joined with S_2 with the results being placed in the third sample relation, denoted as S_3. The combination of all of the sample relations is considered the Correlated Sample Synopsis, or CS2.

3.2 Query Estimation

The process of query estimation is taking the results from a sample query, and using said results to estimate query result sizes.

3.2.1 Source Query Estimation

Source Query Estimation, is the process of estimating query results using sample queries that includes the source relation. Referring back to Figure 7, a source query would be considered a join of relations S_1^* and S_2, or a join between relations S_1^* and S_3. The results of these joins could then be used to estimate the join query size of joins between R_1, and R_2, as well as R_1 and R_3.

Note that the queries used in experiment research used CS2 Source Query Estimation. The use of CS2 for experiment purposes was for query estimation. No-Source Query Estimation is beyond the scope of this thesis. But, it is important to know that CS2 can use No-Source Query (discussed in next section) to not only get query size estimation, but also approximate query processing. AQP includes more aggregate queries such as SUM and AVG.

3.2.2 No-Source Query Estimation

No-Source Query Estimation, is the process of estimating query results using sample queries that do not include the source relation. Again, referring back to Figure 7, a join of S_2 and S_3 would be considered a No-Source Query. In this situation, the relation with the least index, based on the sampling order, would be considered the highest relation. Due to the conditions of a No-Source query not containing a SRSWOR based off the source relation, additional steps must be taken for accurate estimation. Joinable Tuple Sampled Ratio, or JR, is a procedure of backtracking to the source relation in a no-source query (reverse sampling), and supplying it with the ability to estimate the join query size.

4 Experiment

4.1 Setup

For this experiment, a cluster of five nodes on a remote server were created in the Sarah Cloud created by YSU Data Lab[1]. This cluster consists of two nodes being masters and three nodes taking the responsibilities of workers. Master Node One has four Intel Xeon CPU's (E5-2630 v4 @ 2.20 GHz) and 16GB of RAM. Master Node Two has two Intel Xeon CPU's (E5-2630 v4 @ 2.20 GHz) and 10GB of RAM. All worker nodes consist of the same setup, a Intel Xeon CPU (E5-2630 v4 @ 2.20 GHz) processor and 8GB of RAM.

The cluster is running Apache Hadoop with Apache Hive command line setup. The cluster is connected to remotely using Putty SSH, through Cisco AnyConnect VPN Connector.

Two datasets are used, both datasets are generated using TPC-H benchmark[22]. The first dataset created has a total size of 1GB. The second dataset created has a total size of 10GB. Each dataset holds eight relations. The relations are Lineitem, Customer, Orders, Partsupp, Part, Supplier, Nation, and Region.

[1]http://datalab.ysu.edu

4.2 Experiment Setup

The following steps are taken to prepare for experimentation on the big data.

Step 1. Using the source dataset, a source relation as well as a join graph path must be decided on. Figure 8 shows the results of these decisions. Lineitem holds the most many-to-one relationships, and is selected as the source relation. The path that was chosen based on relationships was as follows:

- Lineitem -> Orders, Lineitem -> Partsupp, Orders -> Customer, Partsupp -> Part, Partsupp -> Supplier, Customer -> Nation, Nation -> Region

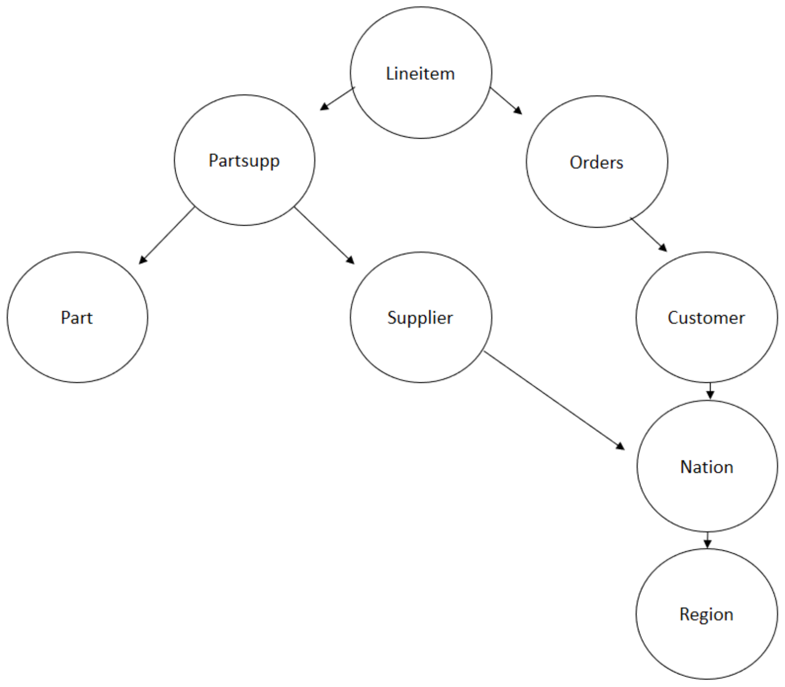


Figure 8: A Join Graph of the TPC-H Dataset

Note that Figure 8 shows multiple paths to Nation. Supplier as well as Customer, or even both, can be used as a join relation for Nation. For these experiments Customer is chosen.

Step 2. A empty set must be created to store the samples of the source dataset. The 1GB and 10GB datasets were denoted as tpch1g and tpch10g respectively. The sample datasets were denoted as s_tpch1g and s_tpch10g respectively.

Step 3. Before creating the SRSWOR a sample dataset size must be selected. The decision was made that the sample dataset size would be one percent of the source dataset. The HQL to create the SRSWOR is as follows:

-
  ```
  create table s_tpch10g.lineitem as select * from tpch10g.lineitem
  where rand () <= 0.01
  Distribute by rand ()
  Sort by rand ();
  ```

The HQL lines Distribute by rand (), and Sort by rand () were added to create a higher rate of randomness. Distribute by rand () takes the entire set of tuples from a table, and distributes them randomly to different reducers. Sort by rand () then takes these sets of random tuples and sorts them randomly on each reducer.

Step 4. Using the created SRSWOR, and following the join graph path, the rest of the sample relations are constructed. The HQL code that was used to create the sample relations are located in Appendix A.

4.3 Results

Overall, a total of 15 queries were tested five times each, over both the 1GB and 10GB source dataset, as well as the 1GB and 10GB sample dataset. The exact queries used in this process are located in Appendix B.

The original dataset, is denoted as the “source” on all graphs, and the correlated sample dataset, is denoted as “sample”. Discussion about the results for the 1GB dataset will be presented first, followed by the 10GB dataset results, and finishing with discoveries made through the testing phase. The datasets were tested on speed of queries, as well as accuracy of the results.

For accuracy tests, we compare the estimated query results by CS2 ($Q_{\text{estimated}}$) with the ground truth query results from the source database ($Q_{\text{ground\_truth}}$) and calculate the absolute relative error. The formula of the absolute relative error is given by:

$$\text{absolute relative error} = \left| \frac{Q_{\text{ground truth}} - Q_{\text{estimated}}}{Q_{\text{ground truth}}} \right| \times 100\%$$

Table 1: Average Times of 1GB Query Search Length (Seconds)

Type	Average	High	Low
Source	16.26	23.24	11.57
Sample	5.33	7.81	3.71

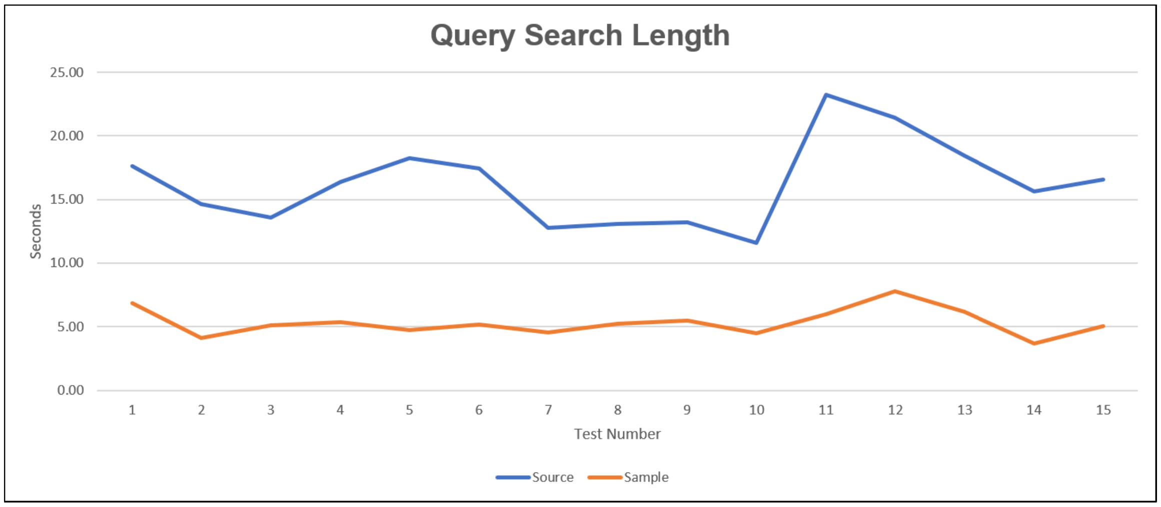


Figure 9: 1GB Dataset Query Search Length Results (Seconds)

The 1GB dataset source dataset averaged a total query time of 16.26 seconds, with a high average of 23.24 seconds in Query 11, and a low average of 11.57 seconds in Query 10. The 1GB sample dataset averaged 5.33 seconds, with a high average of 7.81 on Query 12, and a low average of 3.71 on Query 14. The average speed up from the sample, over the source would be a 205%. The largest speed up was Query 14 at 321.71%, and the lowest speed up was Query 9 at 141.65%. The results were impressive on the 1GB dataset with the sample dataset processing much faster than the source.

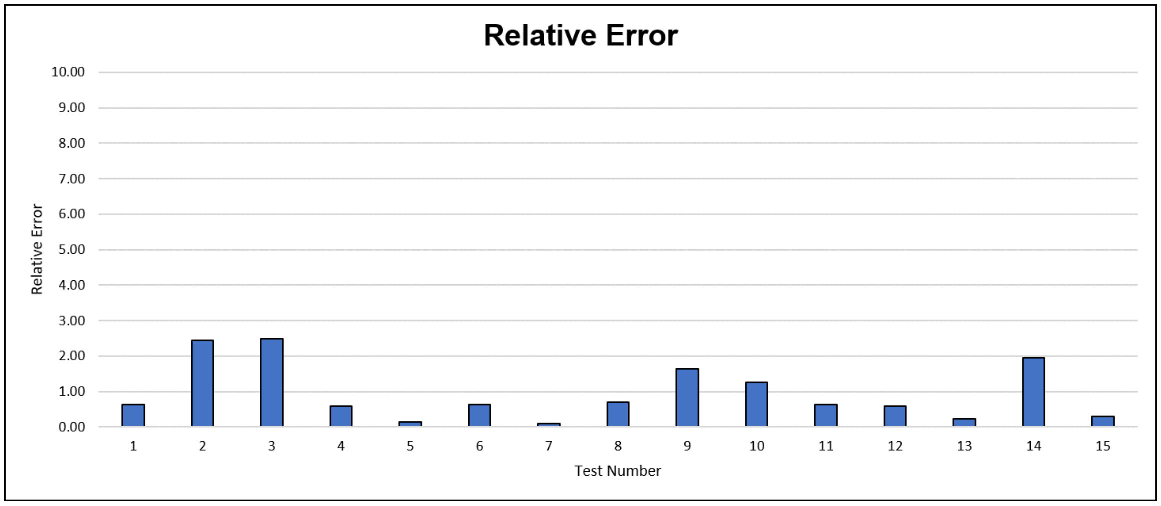


Figure 10: 1GB Dataset Relative Error Results (%)

The average count of tuples for the source dataset was 2,520,952. The average count of tuples for the sample dataset was 25,385. The average join estimation results based off source join estimation was 2,538,533. The average Relative Error for the 1GB dataset was .96%. The highest relative error was 2.50% on Query 3 with the source dataset holding 592,794 tuples, the sample dataset holding 6,076 tuples and source join estimation showing 607,600 tuples. The lowest relative error was .09% on query 7, with the source dataset showing 24,877, the sample dataset showing 249, and the source join estimation showing 24,900.

Table 2: Average Times of 10GB Query Search Length (Seconds)

Type	Average	High	Low
Source	437.49	1692.87	69.57
Sample	9.53	14.84	7.00

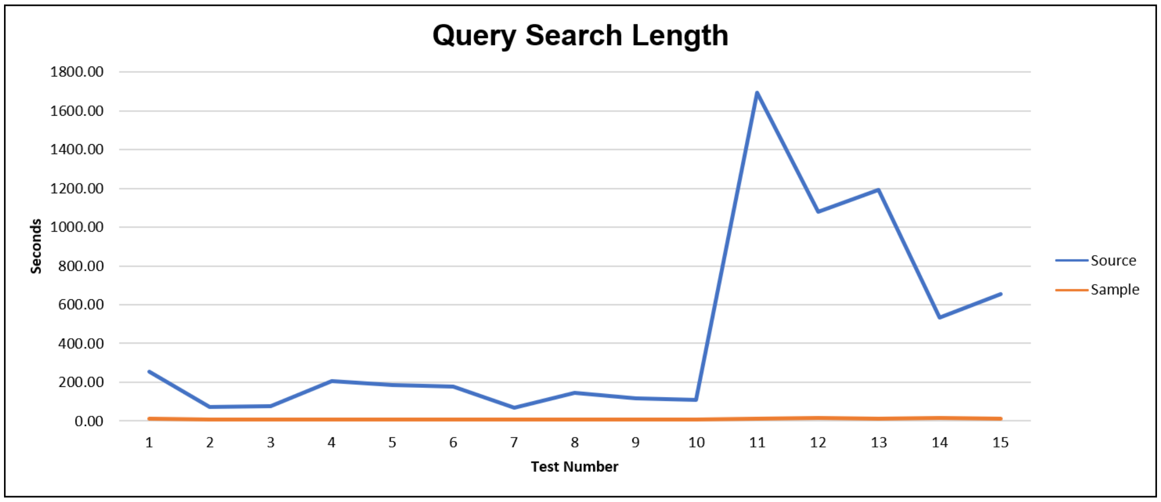


Figure 11: 10GB Dataset Query Search Length (Seconds)

The 10GB dataset source dataset averaged a total query time of 437.49 seconds, with a high average of 1,692.87 seconds in Query 11, and a low average of 69.57 seconds in Query 7. The 1GB sample dataset averaged 9.53 seconds, with a high average of 14.84 on Query 11, and a low average of 7 on Query 3. The average speed up from the sample, over the source would be a 4,489.44%. The largest speed up was Query 11 at 16,071.90%, and the lowest speed up was Query 2 at 758.39%. This shows that the larger the dataset, the better CS2 performs in speed. These results also showed something that was unexpected and will be discussed under Figure 13, Two-Relation Join Query Results.

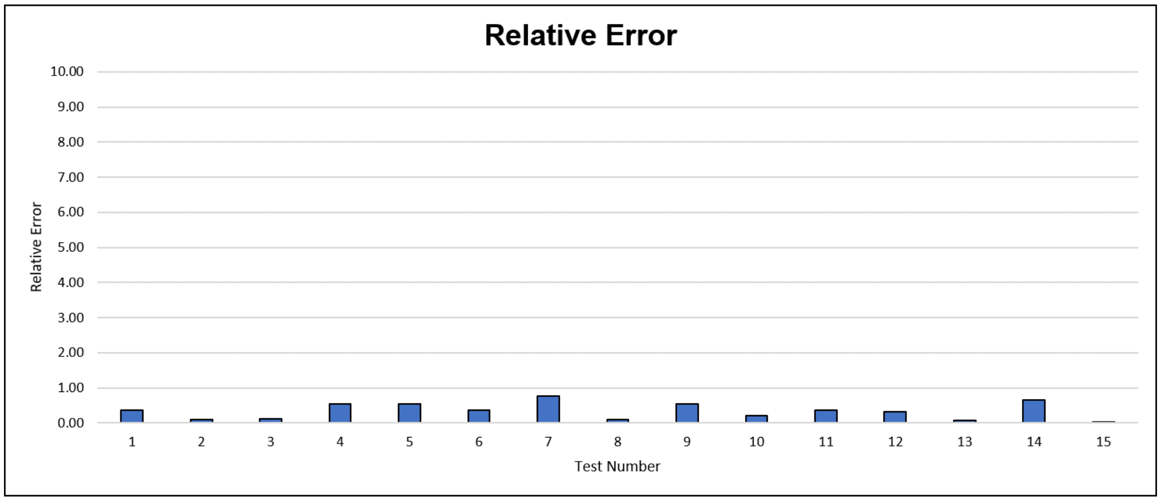


Figure 12: 10 GB Dataset Relative Error Results (%)

The average count of tuples for the source dataset was 25,208,072. The average count of tuples for the sample dataset was 251,168. The average join estimation results based off source join estimation was 25,116,820. The average Relative Error for the 10GB dataset was .34%. The highest relative error was .76% on Query 7 with the source dataset holding 248,493 tuples, the sample dataset holding 2,466 tuples and source join estimation showing 246,600 tuples. The lowest relative error was .01% on query 15, with the source dataset showing 6,047,718, the sample dataset showing 60,474, and the source join estimation showing 6,047,400. The results show that not only does CS2 speed up the larger the data gets, but its accuracy also improves.

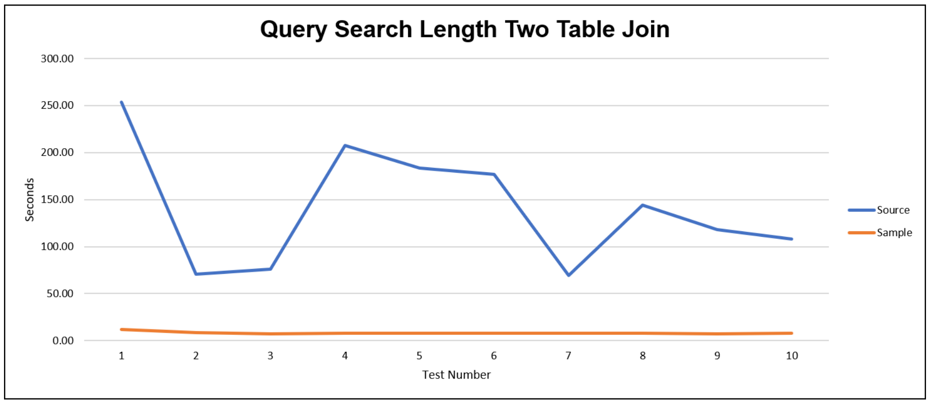


Figure 13: 10GB Dataset Two Relation Join Query Results (Seconds)

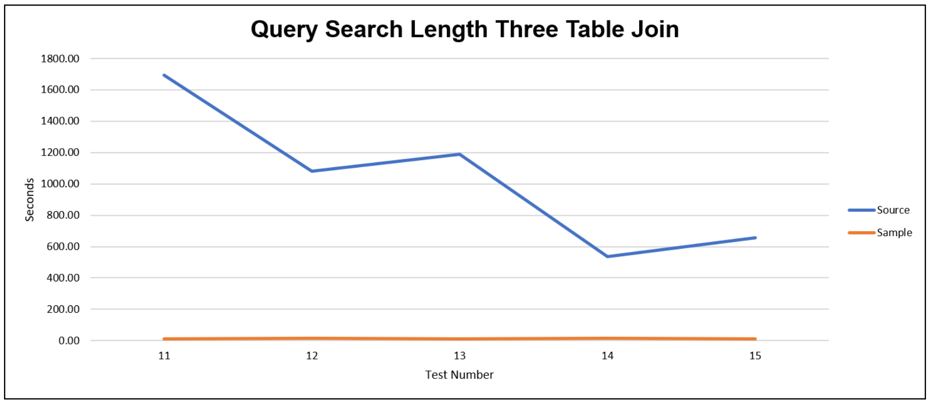


Figure 14: 10GB Dataset Three Relation Join Query Results (Seconds)

After finishing both the 1GB dataset as well as the 10GB database, something unique was recognized. Not only does CS2 speed up the join queries, but when moving from a two relation join, to a three relation join, the time increase is very minimal for CS2. In Figure 13, the two relation join query results, CS2 holds at about a seven second average while the source averages around 180 seconds. When the queries switched to a three relation join in Figure 14, the average for CS2 bumps up to about 10 seconds, while the source relation explodes and averages about 1100 seconds per query.

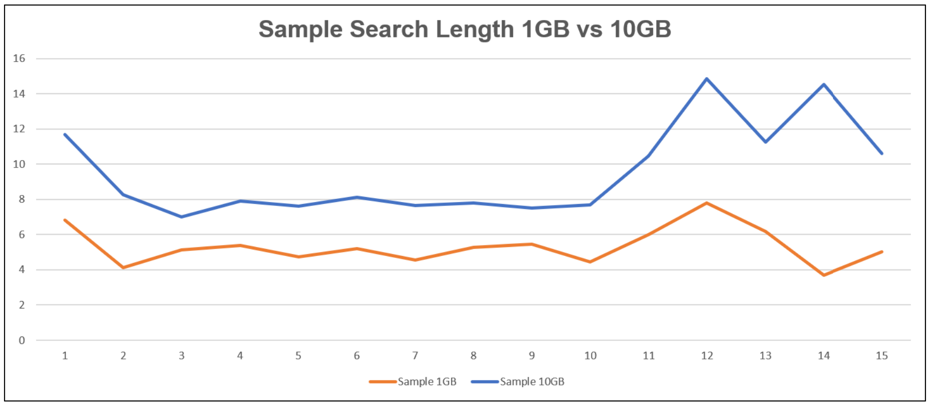


Figure 15: Sample Search Length 1GB vs 10GB Results (Seconds)

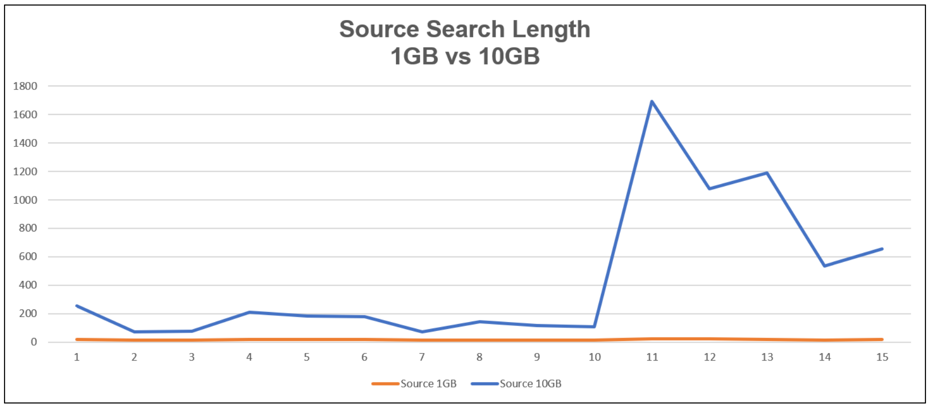


Figure 16: Source Search Length 1GB vs 10GB Results (Seconds)

Figure 15 and 16 show how CS2 maintains a fast query search length as the dataset gets larger. Figure 15 shows the CS2 search length comparison between the 1GB and 10GB datasets. It maintains its form from the 1GB dataset to the 10GB set, only slightly changing on Query 14. In comparison the source dataset search length jumps all over the chart, and when it reaches the three relation join the query length substantially increases.

5 Conclusion and Future Works

In this research, the use of CS2 on big data was introduced. It was discovered that not only does CS2 maintain the accuracy of tuples from its samples in join query estimation, but increases in precision as the dataset grows larger. CS2 also maintained a constant speed and did not increase much as the datasets expanded in size. When the source relation query search length ballooned in size with the three relation joins, CS2 continued do produce low search query lengths. Based off the results CS2 proved to be successful in query optimization and a more efficient alternative to other optimizers such as Join Synopses, in regards to storage requirements.

The type of join estimator that was used for this research was the CS2 Source Estimator. Future research will seek to use No-Source estimator with JR proving that CS2 will also excel with AQP, ultimately providing the ability to accurately estimate customer trends, among other business analytics, at a fraction of the query processing time.

References

[1] "Number of customer transactions home depot/lowe's worldwide 2011-2017 | statistic." [Online]. Available: https://www.statista.com/statistics/318849/number-of-customer-transactions-at-the-home-depot-and-lowe-s-worldwide/

[2] F. Yu, W.-C. Hou, C. Luo, D. Che, and M. Zhu, "Cs2: A new database synopsis for query estimation," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13. New York, NY, USA: ACM, 2013, pp. 469–480.

[3] B. Marr, "How much data do we create every day? the mind-blowing stats everyone should read," Jul 2018. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#181cc81560ba

[4] Y. E. Ioannidis, "Query optimization," *ACM Computing Surveys (CSUR)*, vol. 28, no. 1, pp. 121–123, 1996.

[5] M. Van Rijmenam, "A short history of big data," 2015. [Online]. Available: https://datafloq.com/read/big-data-history/239

[6] A. Gandomi and M. Haider, "Beyond the hype: Big data concepts, methods, and analytics," *International Journal of Information Management*, vol. 35, no. 2, pp. 137–144, 2015.

[7] D. Laney, “3d data management: Controlling data volume, velocity and variety,” *META group research note*, vol. 6, no. 70, p. 1, 2001.

[8] “The four v’s of big data.” [Online]. Available: https://www.ibmbigdatahub.com/infographic/four-vs-big-data

[9] J. Cano, “The v’s of big data: Velocity, volume, value, variety, and veracity,” Mar 2014. [Online]. Available: https://www.xsnet.com/blog/bid/205405/the-v-s-of-big-data-velocity-volume-value-variety-and-veracity

[10] “Hadoop vs hive - find out the best differences,” Oct 2018. [Online]. Available: https://www.educba.com/hadoop-vs-hive/

[11] T. White, *Hadoop: The definitive guide.* " O’Reilly Media, Inc.", 2012.

[12] D. Borthakur, “The hadoop distributed file system: Architecture and design,” *Hadoop Project Website*, vol. 11, no. 2007, p. 21, 2007.

[13] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth *et al.*, “Apache hadoop yarn: Yet another resource negotiator,” in *Proceedings of the 4th annual Symposium on Cloud Computing.* ACM, 2013, p. 5.

[14] S. P. Bappalige, “An introduction to apache hadoop for big data.” [Online]. Available: https://opensource.com/life/14/8/intro-apache-hadoop-big-data

[15] A. Shenoy, *Hadoop Explained.* Packt Publishing Ltd, 2014.

[16] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: a warehousing solution over a map-reduce framework," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, 2009.

[17] A. Swami, "Optimization of large join queries: combining heuristics and combinatorial techniques," in *ACM SIGMOD Record*, vol. 18, no. 2. ACM, 1989, pp. 367–376.

[18] W.-C. Hou, G. Ozsoyoglu, and B. K. Taneja, "Statistical estimators for relational algebra expressions," in *Proceedings of the seventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems.* ACM, 1988, pp. 276–287.

[19] F. Olken, "Random sampling from databases," Ph.D. dissertation, University of California, Berkeley, 1993.

[20] S. Chaudhuri, R. Motwani, and V. Narasayya, "On random sampling over joins," in *ACM SIGMOD Record*, vol. 28, no. 2. ACM, 1999, pp. 263–274.

[21] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy, "Join synopses for approximate query answering," *SIGMOD Rec.*, vol. 28, no. 2, pp. 275–286, Jun. 1999.

[22] T. P. P. Council, "Tpc-h benchmark specification," *Published at http://www. tcp. org/hspec. html*, vol. 21, pp. 592–603, 2008.

Appendix A Correlated Sampling HQL Code

Creating Sample Partsupp Table

- `create table s_tpch10g.partsupp as select distinct partsupp.ps_partkey, partsupp.ps_suppkey, partsupp.ps_availqty, partsupp.ps_supplycost, partsupp.ps_comment from tpch10g.partsupp join s_tpch10g.lineitem on partsupp.ps_partkey = lineitem.l_partkey and partsupp.ps_suppkey = lineitem.l_suppkey;`

Creating Sample Orders Table

- `create table s_tpch10g.orders as select distinct o_orderkey, o_custkey, o_orderstatus,o_totalprice,o_orderdate,o_orderpriority, o_clerk, o_shippriority,o_comment from tpch10g.orders join s_tpch10g.lineitem on orders.o_orderkey = lineitem.l_orderkey;`

Creating Sample Part Table

- `create table s_tpch10g.part as select distinct p_partkey, p_name, p_mfgr, p_brand, p_type, p_size, p_container, p_retailprice, p_comment from tpch10g.part join s_tpch10g.partsupp on part.p_partkey = partsupp.ps_partkey;`

Creating Sample Supplier Table

- `create table s_tpch10g.supplier as select distinct s_suppkey, s_name, s_address,s_nationkey,s_phone, s_acctbal,s_comment from tpch10g.supplier join s_tpch10g.partsupp on supplier.s_suppkey = partsupp.ps_suppkey;`

Creating Sample Customer Table

- `create table s_tpch10g.customer as select distinct c_custkey,c_name ,c_address,c_nationkey,c_phone, c_acctbal,c_mktsegment,c_comment from tpch10g.customer join s_tpch10g.orders on customer.c_custkey = orders.o_custkey;`

Creating Sample Nation Table

- `create table s_tpch10g.nation as select distinct n_nationkey, n_name, n_regionkey,n_comment from tpch10g.nation join s_tpch10g.customer on nation.n_nationkey = customer.c_nationkey;`

Creating Sample Region Table

- `create table s_tpch10g.region as select distinct r_regionkey, r_name, r_comment from tpch10g.region join s_tpch10g.nation on region.r_regionkey = nation.n_regionkey;`

Appendix B Testing Queries HQL Code

Query 1

- `select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey;`

Query 2

- `select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey and ps_availqty > 9000;`

Query 3

- `select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey and ps_supplycost < 100;`

Query 4

- `select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey and l_quantity >= 20;`

Query 5

- `select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey and l_extendedprice >= 40000;`

Query 6

- `select count (*) from lineitem,orders where l_orderkey = o_orderkey;`

Query 7

- `select count (*) from lineitem,orders where l_orderkey = o_orderkey and o_totalprice >= 400000;`

Query 8

- `select count (*) from lineitem,orders where l_orderkey = o_orderkey and l_quantity < 20;`

Query 9

- `select count (*) from lineitem,orders where l_orderkey = o_orderkey and l_discount = .04;`

Query 10

- `select count (*) from lineitem,orders where l_orderkey = o_orderkey and l_shipmode = 'AIR';`

Query 11

- select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey;

Query 12

- select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey and c_acctbal > 500;

Query 13

- select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey and c_mktsegment = 'AUTOMOBILE';

Query 14

- select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey and l_returnflag = 'A';

Query 15

- select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey and l_returnflag = 'N' and c_mktsegment = "HOUSEHOLD";