Random sampling in Apache Hive

Sai Sree Parvathaneni
Iowa State University

Random sampling in Apache Hive

by

SaiSree Parvathaneni

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering (Software Systems)

Iowa State University

Ames, Iowa

2019

DEDICATION

I would like to dedicate this thesis to my family and friends without whose support I would not have been able to complete this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank my major professor, Dr. Srikanta Tirthapura for his guidance, support and encouragement throughout the course of this research and the writing of this thesis. I would like to thank my committee members, Dr. Goce Trajcevski, and Dr. Mai Zheng, for their support and valuable feedback. Next, I would like to thank all professors who taught me classes and helped me to gain immense knowledge in big data analytics.

I would also like to thank graduate department faculty, and staff of Electrical and Computer Engineering department for all their help. Last, but not the least, I would like to thank all my friends at Iowa State University for making my time at Iowa State University a memorable experience.

ABSTRACT

Data generated by humans and machines is growing at a rapid pace. Analyzing the data provides trends, patterns, and useful insights in data which helps to make important organizational decisions [1]. Traditional database systems have been storing and analyzing large amounts of data for many decades. In traditional databases, handling and analyzing growing data needs lots of resources and time. Reading and writing large data from a single disk is significantly slow. Storing and reading from multiple disks and combining them for analyzing on a single CPU is also not reasonable for huge amounts of data [2]. The problem of storing and analyzing large amount of data is handled by Apache Hadoop. Apache Hadoop is a collection of open source big data software's that can efficiently handle storing large amounts of data by dividing data into small blocks and replicates the data to handle system failures. Data is analyzed based on the concept of parallel computation [3]. Hive is a data warehousing software that works on top of Hadoop file system. It has an Hive QL interface to execute queries, and are automatically converted into map reduce or tez or spark jobs.

"Many data mining applications and statistical analysis techniques can use a sample of the data requested in the SQL query without compromising the results of the analysis [4]". For aggregate queries like AVG, SUM, count e.tc., and for analyzing trends in data, sampling gives good approximation about overall data [5]. Analyzing sample population can be achieved with limited amount of resources. There are different sampling techniques to draw sample from a population, and choice of sampling technique depends on type of analysis we perform to achieve the goal. In this thesis, we have investigated different techniques to perform random sampling in Hive. First, we describe about various random sampling techniques and their advantages. Next, we try to understand Hive and its architecture to efficiently store data on Hive. And then, we discuss about the existing techniques to sample the data from traditional database systems as well as Hive. Last, we discuss

about efficient way to get random sample data set for different data set sizes using hive architecture and compare with existing methods.

CHAPTER 1. RANDOM SAMPLING

1.1 Introduction

Data is growing exponentially in fields like social media networking, transportation, science experiments like genome sequencing e.t.c., With growing data, data processing for ad-hoc queries requires lots of resources and time [4]. For approximate query processing to estimate the results, sample data plays an important role. Sample data set is used to perform analysis and draw conclusions for the whole data [6]. Analyzing subset of a large amount data can be achieved with significantly less amount of resources and time.“Instead of completely processing a database query and then sampling the result, we can, in effect, interchange the sampling and query operators, so that we sample prior to query evaluation [8].” In this paper, we focus on random sampling technique. Next few sections describes about various random sampling techniques.

1.2 Simple Random Sampling without Replacement

Simple random sampling without replacement is a random sampling technique in which each sample element of the population has equal chances to get included in the sample. Due to not having any preference in selecting each element in a sample data set, this method gives good representation of the whole data [6].

1.3 Simple Random Sampling with Replacement

Simple random sampling with replacement is a sampling technique in which each element of the sample is chosen using simple random sampling from whole population. Once a sample element is chosen using simple random sampling from whole population, we make a copy of the sample and put it back in the population for selecting another sample element from whole population. Each element is equally likely to get selected. Duplicates are allowed [6].

1.4 Bernoulli Sampling

Selection of elements to make a sample is determined by applying independent Bernoulli trial to each element of the population. Every element of the population is selected with an equal probability. It is a fixed percent sampling. To select a sample of percent p, independent Bernoulli trial is applied on each element, where each element is selected with a probability of p/100. No duplicates are allowed [5]. Randomly selecting the sample with no bias made it a good sampling technique to represent large population [9].

CHAPTER 2. APACHE HIVE

Traditional databases holds large amounts of data. In traditional databases, data is processed on a single CPU, and computations are performed one after another. As a result reading and writing data from a single disk is a slow process. It is not resonable to read, write and process large amounts of data like terra bytes of data from a single disk with limited resources and time [7]. Apache Hadoop is a open source framework that can handle large data storage and parallel processing of computations. Hadoop handles large data storage by dividing the data into small data blocks and stores them on different nodes, and process the data in parallel using a concept called MapReduce [3]. Map function collects the data from each data block location, and breaks the data into key/value pairs. They are transferred to reducers for further processing of whole data [11]. Hive is a data warehousing software, which works on top of Hadoop file system. Hive has a SQL interface to execute Hive QL queries which are converted to map reduce jobs [12]. Figure 2.1 shows relation between HDFS, MapReduce and Hive.

2.1 Hive Architecture

HiveQL queries scans the entire data to execute a query. When data gets larger, the cost of processing also increases. Hive supports two ways Partitioning and Bucketing to divide and manage the data into small blocks based on column values. It helps to easily manage the data and optimize the query processing when the data is huge.

2.1.1 Partitioning in Hive

In partitioning, data is divided into partitions based on specific predefined columns. All the data having same column values goes to same partition. Data blocks are divided and stored in sub directories in Hadoop Distributed File System based on the partitioned columns. When a query

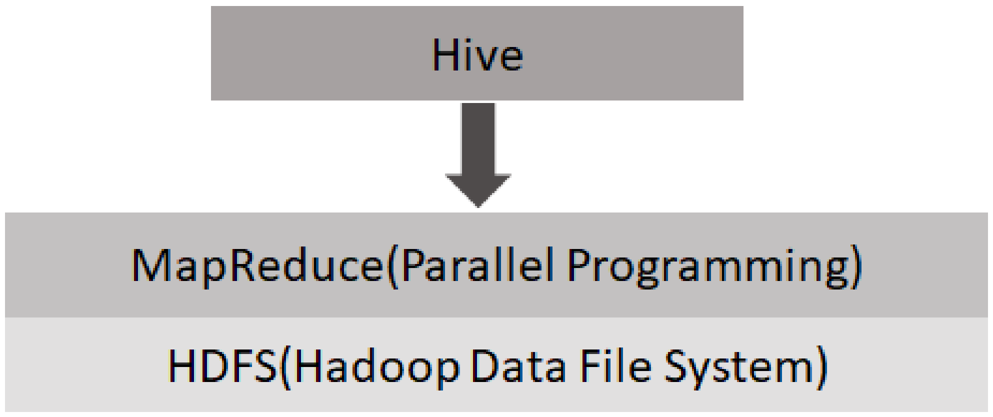

Hive
MapReduce(Parallel Programming)
HDFS(Hadoop Data File System)

Figure 2.1 Hadoop HDFS, MapReduce and Hive

is executed on a specific partitioned column, hive gets the data from the partitioned sub directory without scanning the whole table. This cuts down the resources and time for processing the query [12]. Query1 is a sample query to create a partitioned table StudentDetails partitioned by year. For every different value of year, a partition is created.

Query 1 : To create a partitioned table StudentDetails partitioned by year

```
CREATE TABLE StudentDetails( StudentID INT, Name STRING, Year STRING, Dept STRING)
PARTITIONED BY (Year STRING)
```

2.1.2 Bucketing in Hive

In Hive, Bucketing is another technique in which data will be divided into specific number of blocks within a partition or table without a partition to manage large amounts of data and to optimize the query processing. Data is divided into buckets based on specified column values [12]. Data is divided into user defined number of buckets unlike partitions where number of partitions depends on number of different partitioned column values. Data goes into bucket based on hash function and number of buckets specified. Query 2 is a sample query to create a table StudentDetails partitioned by 'year' and data divided into 2 buckets. Figure 2.2 shows sample data divided into buckets and partitions based on predefined partitioned and bucketed columns.

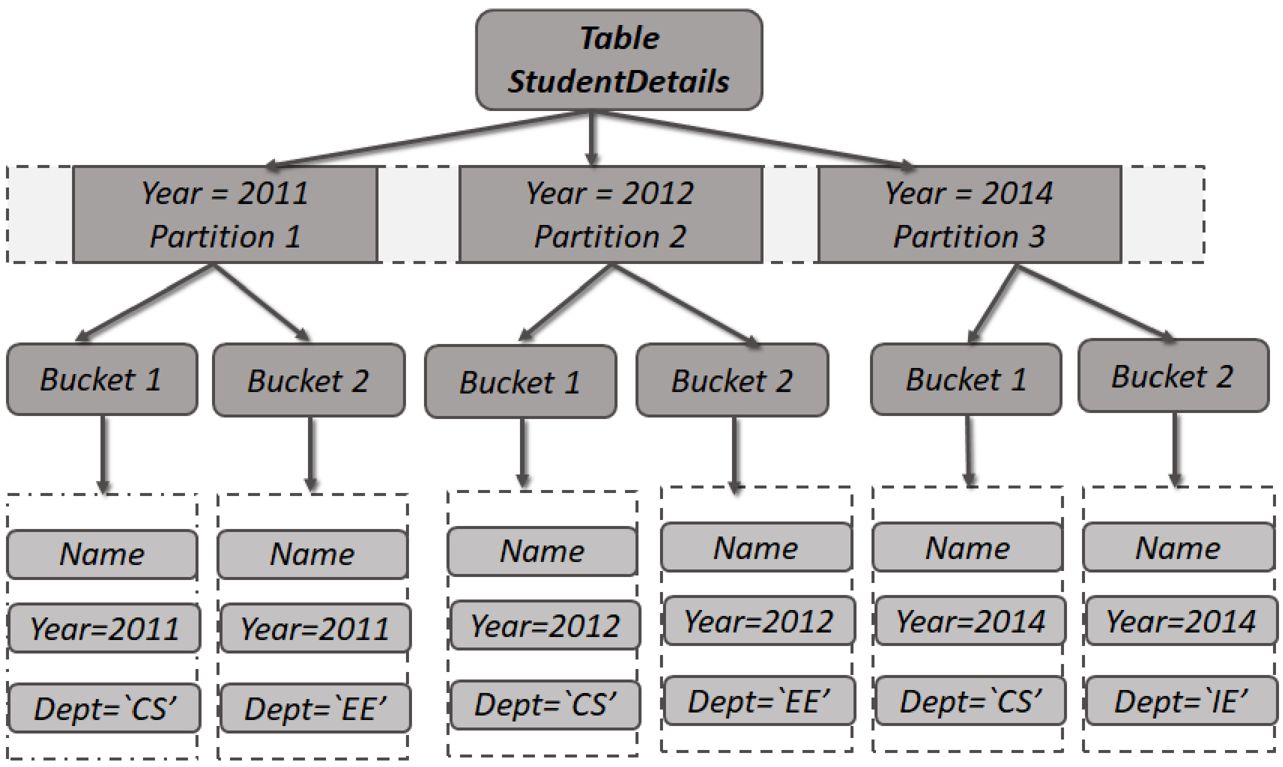


Figure 2.2 Sample data in StudentDetails table divided into partitions and buckets.

Query 2: To create a partitioned table StudentDetails partitioned by year and divided into 2 buckets by dept

CREATE TABLE StudentDetails(StudentID INT, Name STRING, Year STRING,Dept STRING) PARTITIONED BY (Year STRING) CLUSTERD BY (Dept) INTO 2 buckets

CHAPTER 3. REVIEW OF LITERATURE

Simple random sampling operators are available in today's database systems to draw efficient sample datasets. First, we will discuss about the random sampling operators in traditional database systems like relational database and data warehouses which have been used for many decades [2]. Secondly, we will discuss about types of sampling techniques available in hive using Hive QL queries.

3.1 Simple Random Sampling without Replacement in RDBMS

In relational databases like mySQL, a simple random sample can be drawn by SORT and rand() keywords. For example, to draw a simple random sample of size 100 from StudentDetails table can be written as

Query 3 Select 100 random rows from StudentDetails table

```
SELECT * FROM StudentDetails SORT BY rand() limit 100
```

Query 3 scans the whole data and sorts the data randomly and selects top 100 units from the sorted data.

3.2 Bernoulli Sampling in RDBMS

To select a random sample of P percent of data can be achieved with row level Bernoulli's sampling using rand() keyword. For example, to draw a simple random sample of P percent of data can be achieved from following query.

Query 4 Select P percent of rows from StudentDetails table

```
SELECT * FROM StudentDetails rand() <= P/100
```

Query 4 scans each record in the table, and assigns each record a random value between 0 and 1, and gives out the resulting records that has assigned number less than P/100. Each record gets selected with a probability of P. This query gets approximately P percent simple random sample of total data. It does not provide the exact number of sample size. It gives less or more samples than P percent of data. This type of sampling works best when the data is indexed [5].

In traditional database systems, all the data is processed in series. The cost of processing a query is more when the data is huge. To achieve a simple random sample from large amount of data needs lots of resources and time. In order to handle large amount of data, RDBMS needs more CPU's or more memory to perform efficiently.

3.3 Types of Sampling in Hive

In the following sections, we discuss about three sampling techniques that are available in Hive.

3.3.1 Random Sampling

In Hive, random sampling can be done using 'distribute' and 'sort by rand()' keywords.Below is the query that is executed to get a random sample of size 100 from table.

Query 5 Select 100 random rows from StudentDetails table in Hive

```
SELECT * FROM StudentDetails DISTRIBUTE BY rand() SORT BY rand() limit 100
```

In Query 5, 'distribute by rand()' keyword in the query distributes the data among mappers randomly, and 'sort by rand()' keyword sorts the data in each reducer randomly. All the data get sorted randomly, and the limit helps to get the sorted random sample. This query on hive does the whole table scan [12].

3.3.2 Bucketing Sampling

Bucketing is a technique to divide the data into desired number of blocks. TABLE SAMPLE is a clause that helps to select desired number of buckets from the available buckets, based on the clustered column.

Query 6 Select x out of y buckets clustered by colname in StudentDetails table in Hive

```
SELECT * FROM StudentDetails TABLESAMPLE(BUCKET x OUT OF y [ON colname])
```

In Query 6, column name can be any column in a table. Data will be clustered by the column name and the elements of a table are divided into y number of buckets [12]. The cost of processing is high for large amount of data as the table is divided into buckets in memory and gives sample data. It is hard to assume x and y values for a specific sample size data.

3.3.3 Block Sampling

This type of sampling is used to get specific size block of data or rows of data from hive.

Query 7 Select n percent of StudentDetails table in Hive

```
SELECT * FROM StudentDetails TABLESAMPLE( n PERCENT)
```

Query 7, gets n percent of the data.There is a chance of selecting whole data in case of query failure. This query doesn't ensure required number of rows for sample selection, and also the data that is drawn is top rows. It cannot achieve random sampled data of required size [12].

Query 8 Select x out of y buckets clustered by colname in StudentDetails table in Hive

```
SELECT * FROM   StudentDetails TABLESAMPLE(n rows)
```

In Query 8, It draws top n rows in data block. This type of sampling doesn't handle random sampling [12].

CHAPTER 4. ALGORITHMS AND ANALYSIS

4.1 Introduction

In chapter three, we have discussed various methods to perform sampling in RDBMS(Relational Database Management System) and Hive. In this section, we describe different algorithms for random sampling in Hive. First, we discuss an algorithm which sorts the whole data randomly and selects random sample of specific size. Second, we discuss the algorithm to select the random sample data set using Bernoulli's Sampling. Third, we discuss our algorithm to get random sample data set using bucketing in hive to store data for optimal query processing. From all the three algorithms we expect an output of random sample dataset of exactly specific size.

4.2 Simple Random Sampling using Sorting

Suppose we want to draw a sample of size S from a table T. Let the total number of rows in the whole table be n. Below is the algorithm to select a random sample of size S from table by sorting the whole data.

Algorithm 1 SortSRS(T,S)

```
1. Select whole data from T
2. Distribute the data among all reducers randomly.
3. Sort the data randomly
4. Limit the selected and sorted data to S
```

The method described in Algorithm 1 scans the whole data, distribute the data among all the reducers randomly, and sorts the data randomly in each reducer, and finally outputs required sample size.

4.3 Random Sampling using Bernoulli's Sampling

The following section describes the algorithm to get a random sample of exactly specific size using Bernoulli's Sampling. Applying Bernoulli sampling on each row selects a P percent of data, by selecting each row with a equal probability of P/100 [5].The sample size achieved is approximate, and sometimes it may be less than required sample size [5]. To achieve required sample size, we select 5 percent more data expecting it to get sample of atleast required size, randomly sort the data, and select required samples. Suppose we want to draw a sample of size s from a table T with row count of n. Since the data is selected at random, it always give a good random sample. Algorithm 2 describes to select a random sample of size 's' from table using Bernoulli sampling.

Algorithm 2 BerSRS(T,s)

```
1. $count \leftarrow 0$
2. $count = (s / n)+0.05$
3. Select data from T, where $rand() \leq count$
4. Sort the selected data randomly
5. Output the data with limit s
```

Query executed for Algorithm 2 scans the whole table, assigns a random value between 0 and 1 to each row, and selects (s/n)+0.05 probability of data, and then sorts the data to get the sample of required size s.

4.4 Our Algorithm: Random Sampling using Bucketing

In Algorithm 1, entire data in table is sorted to get random sample, which drastically reduces the performance of executing the queries when the data is large. Algorithm 2 performs better than Algorithm 1, because of sorting the approximate sample size rows instead of sorting the whole data. Both algorithms need additional resources when the data and sample sizes are large. In order to avoid whole table scan while sampling, we divide the whole data in table T into buckets(nothing but data blocks) in a new table T_b, where data is randomly shuffled and stored. Hive inbuilt queries can only clusters the data by column name in the table and divides the data into specific number of buckets. So, we have created hive tables T_b with buckets clustered by random number

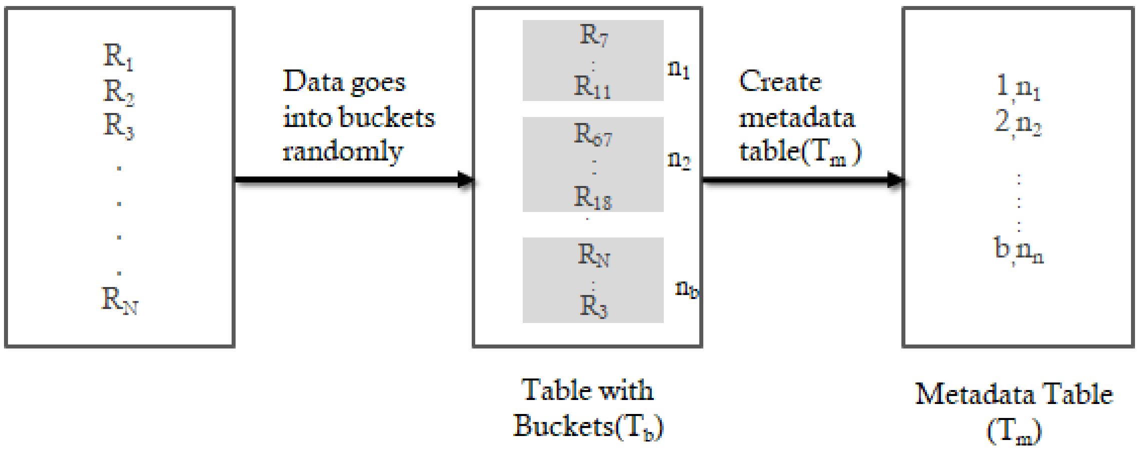


Figure 4.1 ComputeBucketedTable algorithm

in a column which was a randomly generated value between 0 to 1, and is generated using keyword 'rand()'. Data is divided into buckets based on hash value of clustered column and number of buckets. Bucketing in hive tries to divide the data into approximately equal bucket size. As every value in the column is different, the data is distributed randomly among all the buckets. Now the newly created table with buckets has randomly selected data almost equally distributed among specific number of buckets. Let T be table on Hive. We consider a table T_b, with data divided into b number of buckets clustered by a random value. Let i be the bucket number. All the meta data about b number of buckets and count of rows in each bucket b_i in table T_b is stored in a separate table T_m. We can remove the original table T to save space, we can use the T_b as a full table with an additional random column value. Let the number of samples that are required from table be s.

Algorithm 3 ComputeBucketedTable(T,b)

```
1. Select data and add additional column randnum using keyword rand()
2. Create table $T_b$ with b buckets clustered by randnum column. New table $T_b$ has data from
table T randomly distributed into b buckets.
3. Output: Create a metadata table $T_m$ that has count of elements in each bucket.
```

Algorithm 4 BucketingSRS(T_b,s)

```
Require: $s \geq 0$
  Initialize: $count \leftarrow 0$, $selectedbuckets \leftarrow \{\}$,$lastbucket \leftarrow 0$,$limit \leftarrow 0$, $S \leftarrow \{\}$, $L \leftarrow \{\}$
  while $count < s$ do
    Randomly select a bucket $b_i$ without replacement, $i \neq selectedbuckets$
    Find the count of the number of rows in selected bucket $b_i$ from table $T_m$
    $count \leftarrow count+$ sizeof $b_i$
    if $s \geq count + sizeof b_i$ then
      $selectedbuckets \leftarrow selectedbuckets \cup i$
      $S \leftarrow S \cup$ elements of $b_i$
    else
      $limit = s - count$
      $count \leftarrow count + limit$
      $L \leftarrow$ randomly select $limit$ elements from $b_i$ (Bernoulli sampling is used for efficiency)
    end if
  end while
  Return: $S \cup L$
```

CHAPTER 5. EXPERIMENT AND RESULTS

This chapter describes about the experiments we performed to analyze the time of processing for selecting different random sample sizes from different table sizes using algorithms that are discussed in chapter 4. We used Hive 2.3.4 version on Hadoop 3.1.1 to perform our experiments. We used lineitem table the largest table in TPCH Benchmark database as a dataset to carry our experiments. TPCH is one of the industrial benchmark support systems for databases [13]. We generated 'lineitem' tables from small scale to large scale data, ranging from 1 million(128MB) to 320 million rows (41 GB) of table data. We performed all the experiments multiple times and calculated the average of all the time values.

5.1 Bucketing Algorithm On Different Table Sizes

In this section, we compared time of processing to draw a uniform random samples of sizes ranging from 10 to 100M rows for different size tables using our Algorithm 4. Table 5.1 shows the time of processing to create the 16 bucket tables using Algorithm 4, table sizes ranging from 1M to 320M. Time for creating bucketed tables increases with increase in table size and number of buckets. Database tables T with different sizes ranging from 1 million to 320 million rows are used to built tables T_b with 16, 32 and 64 buckets using Algorithm 3. Query 7 is a sample query that is executed to create and load lineitem table of 1 million rows with 64 buckets, and Query 8 to draw a sample of 10000 from table T_b created with Query7.

Query 7 Create and load lineitem table with 64 buckets from original table

```
CREATE TABLE IF NOT EXISTS lineitem_1m_buckets_64( l_suppkey bigint,l_quantity
double,randnum double) CLUSTERED BY (randnum) INTO 64 buckets
STORED AS SEQUENCEFILE;
INSERT OVERWRITE TABLE lineitem_1m_buckets_64 SELECT *,rand() as randnum
```

```
 FROM lineitem_1m
```

Query 8 Draw random sample of size 10000 using our bucketed algorithm

```
SELECT * FROM lineitem_1m_buckets_64 TABLESAMPLE(BUCKET 52 OUT OF 64)
WHERE rand() <= 0.6842762907522517 SORT BY rand() LIMIT 10000
```

Table 5.1 Table shows time taken to execute computeBucketedTable(T, b) algorithm, b = 16

Table Size(Rows)/ Time(Min)	T_b(min)	T_m(min)	$T_b + T_m$(min)
1 Million	0.57	6.74	7.31
20 Million	2.32	8.08	10.40
40 Million	3.42	7.82	11.24
80 Million	7.04	10	17.04
160 Million	12.33	12.75	25.08
320 Million	22.59	14.43	37.02

Figure 5.1 and Figure 5.2 compares the time of processing for different sample sizes, and for different tables sizes, respectively using tables T_b of different size data with 64 buckets. Results from Figures 5.1 shows that for a specific table size data, time for selecting random samples is either constant or steadily increasing with the number of samples selected from the table T_b. As the sample size is increasing, the number of buckets selected from table T_b are either constant or increasing which in turn effecting the processing time. When the sample size is achieved with same number of buckets, then the time of processing is increasing slowly with sample size. Similarly, Figure 5.3 and Figure

5.4 were generated using different size tables T_b with 32 buckets, and Figure 5.5 and Figure 5.6 were generated using different size tables T_b with 16 buckets. From Figures 5.2 , 5.4 and 5.6 for selecting specific size sample, the time of processing increases with increase in table data size. As the table size increases, the number of elements in each bucket increases, which proportionally increases the time taken to select samples from the bucket.

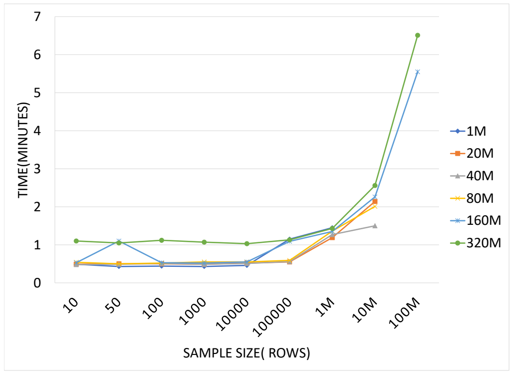


Figure 5.1 Comparison of time of processing and sample size for different table sizes. Each table with 64 buckets.

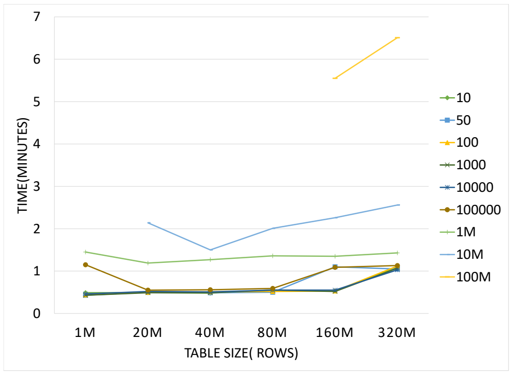


Figure 5.2 Comparison of time of processing and data size for selecting different sample sizes. Each table with 64 buckets.

5.2 Simple Random Sampling using Sorting

Simple random sampling using sorting method scans the whole data in the table to select the elements for scanning, and sort maximum of all elements randomly in the reducers. Figure 5.7 shows the time required for processing to select random samples of different sizes ranging for 10 to 100M. Figure 5.8 shows the time of processing for random sample selection and different table sizes ranging from 1M to 320M. It shows that time of processing increases with increase in table size and sample size. Random sampling for sorting algorithm failed to retrieve samples after 10M for table of size 160M rows, and for sample sizes above 1M for table of size 320M rows. Random sampling using sorting requires more resources to achieve samples when the data is large and sample sizes

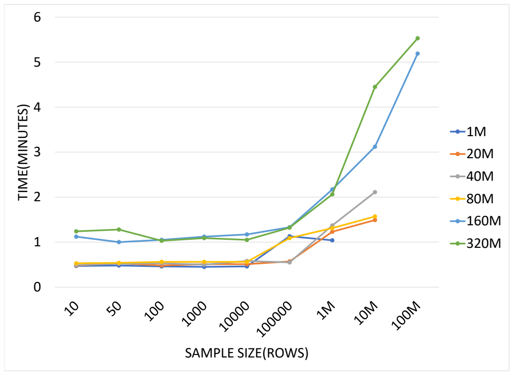


Figure 5.3 Comparison of time of processing and sample size for different table sizes with 32 buckets

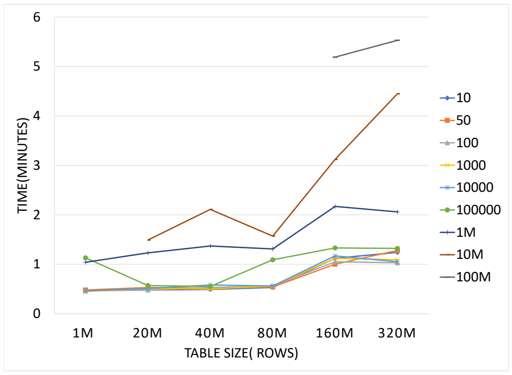


Figure 5.4 Comparison of time of processing with different table sizes for selecting different sample sizes.Each table data is divided into 32 buckets.

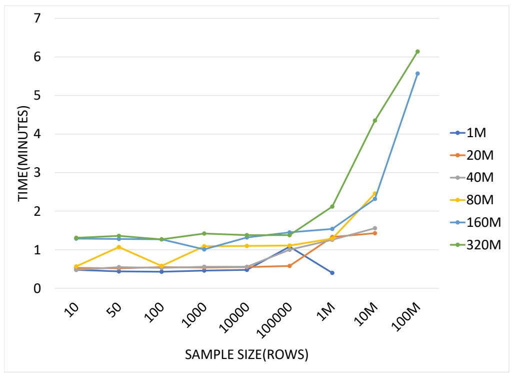
are large. Query 9 is a sample query to draw a sample of size 10000 from lineitem table using simple random sampling using sorting algorithm.

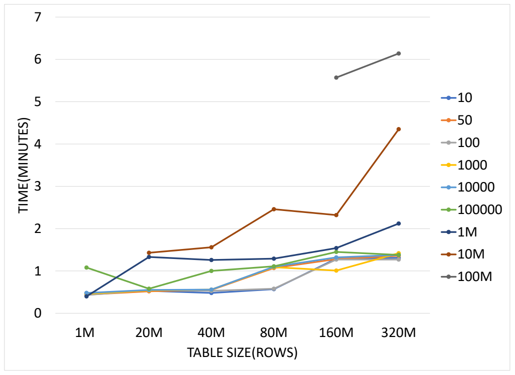
Query 9 Draw random sample of size 10000 by simple random sampling using sorting algorithm

```
SELECT * FROM lineitem SORT BY rand() LIMIT 10000
```

5.3 Bernoulli Sampling

Sorting whole table and selecting random samples using Algorithm 1 needs more resources when the data and sample sizes are large. Our resources are not enough to use Algorithm 1 to select random samples from very large tables. This section shows time taken to select a random sample using Bernoulli's sampling using Algorithm 2 on full table. Figures 5.9 and 5.10 shows the time of processing to select uniform random samples of data ranging from 10 to 100M from tables of sizes ranging from 1M to 320M using Bernoulli sampling. Figure 5.9 shows that for a specific sample size, time of processing is increasing with increase in table size. Figures 5.10 shows time of processing





,

Figure 5.5 Comparison of time of processing and sample sizes for different table sizes with 16 buckets

Figure 5.6 Comparison of time of processing with table sizes for selecting different sample sizes. Each table is divided into 16 buckets.

to select the samples increases with increase in sample size. This method selects samples scanning through the whole table to select approximate percent of samples of required size, which increases the time of processing to select the samples for larger data sets. As the sample size increases, the time taken to sort the approximate percent samples randomly also increases. Query 10 is a sample query to select random samples of size 10000 from lineitem table using Bernoulli sampling.

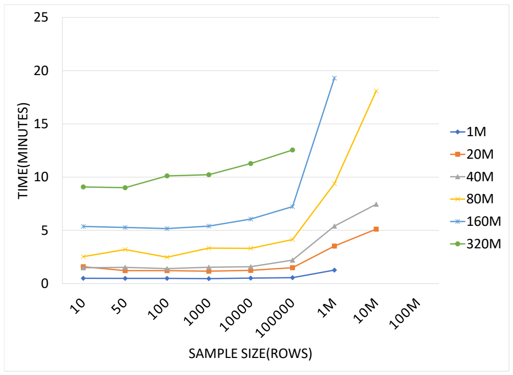
Query 10 Draw random sample of size 10000 using Bernoulli's sampling algorithm

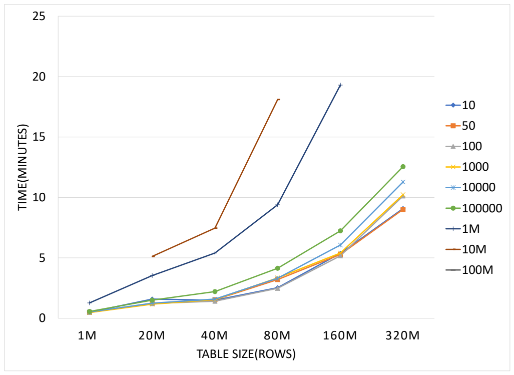
```
SELECT * FROM lineitem WHERE rand() <=0.06

SORT BY rand() LIMIT 10000
```

5.4 Comparing Bucketing Sampling, Sorting and Bernoulli Sampling

In this section, we compared all the three algorithms for selecting sample of different sizes for different table sizes obtained in above sections. Figures 5.11 and 5.12 shows the time of processing for selecting different sample sizes using bucketing algorithm from tables of 16, 32, 64 bucket sizes, simple random sampling using sorting and Bernoulli sampling from full table of size 80M rows. It shows that time of processing to select a sample is lowest using bucketing algorithm,





,

Figure 5.7 Comparison of time of processing to select a random sample using sorting algorithm for different table sizes.

Figure 5.8 Comparison of time of processing and different table sizes for selecting different sample sizes using sorting algorithm on full table.

then Bernoulli sampling and highest using sorting method. Figures 5.13 and 5.14 shows time of processing to select different random samples for different bucket count tables of size 320M rows of data using Bernoulli sampling and bucketing sampling. For large amount of 320M simple random sampling using sorting failed with available resources. Figure 5.13 shows that the time taken to select samples increases with increase in sample size using bucketing algorithm, also it is clear that time taken by Bernoulli sampling without buckets is always higher than sampling using bucketing. More statistical results for other table sizes are included in Appendix.

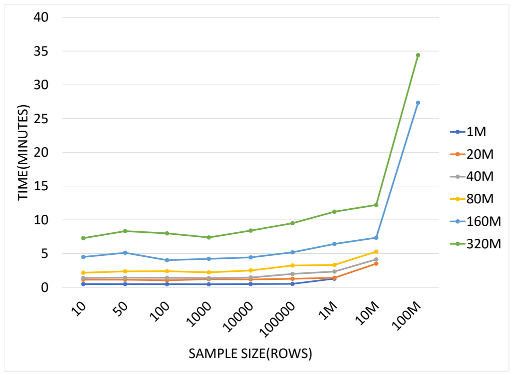


Figure 5.9 Comparison of time of processing and select different sample sizes for tables with different sizes using Bernoulli's sampling.

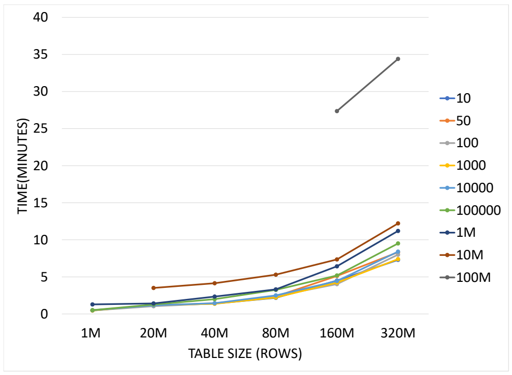


Figure 5.10 Comparison of time of processing to select samples and different table sizes using Bernoulli's sampling.

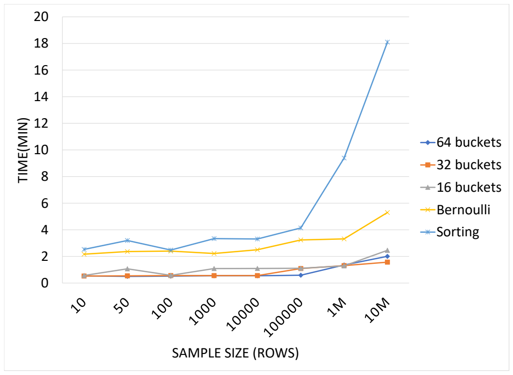


Figure 5.11 Comparison of time of processing and different sample sizes for 80M records table with different algorithms

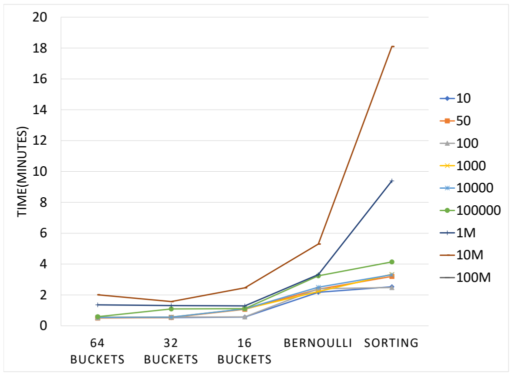


Figure 5.12 Comparison of time of processing and different algorithms to get sample of different sizes from table of 80M rows.

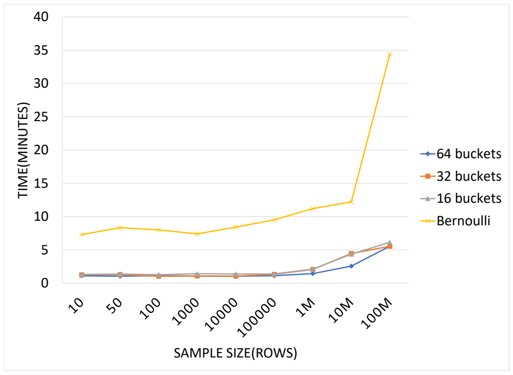


Figure 5.13 Comparison of time of processing and different samples sizes for a different bucket count tables with 320M records.

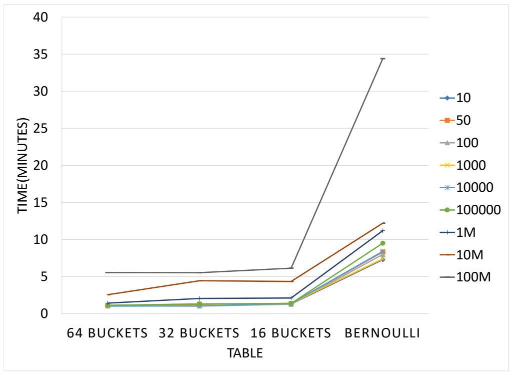


Figure 5.14 Comparison of time of processing and table with different bucket sizes of table with 320M records for different sample sizes.

CHAPTER 6. VERIFICATION OF ACCURACY OF ALGORITHM

In order to verify the accuracy of randomness of samples that are selected, we created a table T with 1 million size using 'lineitem' table from TPCH database. We add a new column called row number which has unique consecutive number from 1 to size of table T. We created bucketed tables T_b of 64, 32 and 16 buckets using T. We used Bernoulli sampling on table T, and Bucketing sampling on T_b to achieve a sample size of 1000, 10000 and 100000.

6.1 Verify by Dividing The Data

We divided 1 million rows into 100 parts, and calculated the count of all the values that are fallen in each part. For example, counted rows in the sample of 10000 that has rows that has number between 0 to 10000, 10001 to 20000 e.t.c. For a good random sample, count of elements in each part should be approximately same size. We calculated average standard deviation of count of elements that are in each part shown in Table 6.1. For a sample size of 1000, we expect approximately 1000/100 elements fall in each part. The standard deviation of count of elements fallen in 100 equal parts is around 3.17 for samples drawn using sorting, 3.26 using Bernoulli sampling, and 2.85, 2.81, and 3.2 using bucketing algorithm. It shows that sampling using bucketing yields a random samples which gives a good representation of the whole population.

6.2 Verify Using Query Approximate Sampling

We have performed Query 11 on the tables having 1K, 10K and 1M samples in section 6.1 to find the approximate average value of quantity column in lineitem table of size 1M. Calculated average error percentage of average quantity for sampled tables. Compared average error from samples obtained by random sampling using sorting, Bernoulli sampling and sampling using bucketing.

Table 6.1 Table shows average standard deviation of count of rows falling into 100 equal parts of 1 million rows.

Sampling Method/ Sample Size	1000	10000	100000
Sorting	3.176	9.56	29.96
Bernoulli	3.26	8.72	32
16 buckets	2.85	10.13	26
32 buckets	2.81	9.36	28.5
64 buckets	3.2	9.17	32

From Figure 6.1 it is clear that all the samples that are drawn using bucketing hold close error percent as Bernoulli sampling and simple random sampling using sorting.

Query 11 Calculate average of each supply key group from lineitem table

```
SELECT l_suppkey, AVG(l_quantity) from lineitem_1m GROUP BY l_suppkey
```

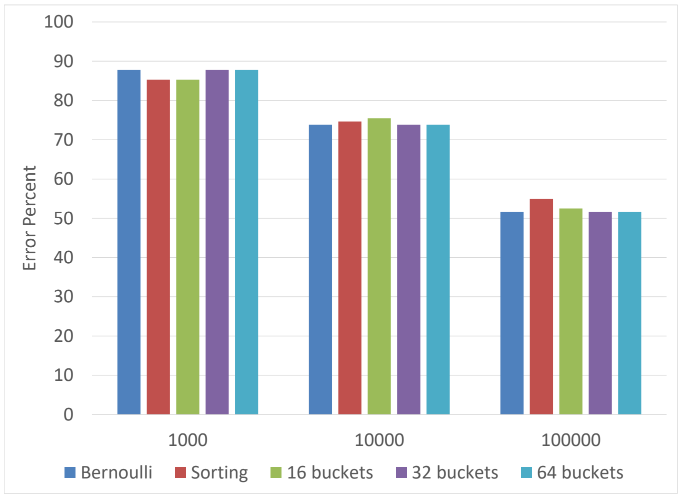


Figure 6.1 Comparison of error percentage and sample sizes of tables generated using Bernoulli sampling, random sampling using sorting and bucketing sampling on different number of buckets

CHAPTER 7. CONCLUSION AND FUTURE WORK

Our algorithm, random sampling using bucketing gives random sample. Less additional storage is required to create a table with buckets of randomly distributed data. Comparing all three methods, sampling using bucketing performs faster than full table scan. When the data size or sample size is small, the difference in the time of processing to select random sample using all three methods is not very high. Considering the time taken to create bucketed table, Bernoulli sampling or random sampling using sorting can be used for small table sizes. We conclude that to draw a random sample of fixed size, sampling by bucketing algorithm significantly performs well when compared to Bernoulli's sampling and random sampling using sorting when the data is huge. In Algorithm 1 and Algorithm 2, in order to get sample data whole table is scanned. Algorithm 1 and Algorithm 2 needs resources proportional to the data size of table and required sample size. Problem of increase in cost of processing with increase in data size and sample size are handled in Algorithm 4. Algorithm 4 using bucketing sampling, the time of selection of samples is slowly increasing with data size of table and sample size. With exponential growth in data generated by devices and individuals, to get data insight using limited resources and time is a challenging task. Future work can include finding efficient ways to handle different sampling techniques using architectural properties of software's to store the data in a logical way for optimal query processing.

BIBLIOGRAPHY

[1] U. Sivaraja, M. Kamal, Z. Irani,V. Weerakkody. Critical analysis of Big Data challenges and analytical methods.In Journal of Business Research,pp. 263-286,2017.
Available: https://www.sciencedirect.com/science/article/pii/S014829631630488X

[2] G. Trujillo, C. Kim, S. Jones, R. Garcia, J. Murray. Virtualizing Hadoop: How to Install, Deploy, and Optimize Hadoop in a Virtualized Architecture.VMware Press, 2015.

[3] Tom White (Tom E.) Cutting, Doug. Hadoop: The definitive guide. OReilly for Higher Education, 2010.

[4] S. Chaudhuri, R. Motwani, V. Narasayya. On Random Sampling over Joins. In ACM SIGMOD Record,pp 263-274, 1999.

[5] Peter J. Haas, Speeding up DB2 UDB Using Sampling. IBM Almaden Research Center, 2003.
Available: http://www.almaden.ibm.com/cs/people/peterh/ idugjbig.pdf

[6] F. Olken. Random Sampling from Databases. Available:`http://db.cs.berkeley.edu/papers/UCB-PhD-olke`

[7] S.Tirthapura, Cpr E 419: Software Tools for Large Scale Data Analysis, 18 Jan. 2018, Iowa State University. Microsoft PowerPointpresentation.

[8] F. Olken, D. Rotem. Random sampling from database : a survey.In Statistics and Computing, pp. 25-42,1995.

[9] G. Depersio. What are the advantages of using a simple random sample to study a larger population.
Available: https://www.investopedia.com/ask/answers/042915/what-are-advantages-using-simple-random-sample-study-larger-population.asp

[10] A. Gandomi, M. Haider.Beyond the hype: Big data concepts, methods, and analytics. In International Journal of Information Management, Volume 35, Issue 2, pp. 137-144, 2015.

[11] J. Dean, S. Ghemawat.MapReduce: Simplified Data Processing on Large Clusters. In Sixth Symposium on Operating System Design and Implementation, pp. 137-150, 2004.

[12] Dayong Du. Apache Hive Essentials. O'Reilly Media, 2015.

[13] TPC-H is a Decision Support Benchmark.
Available:http://www.tpc.org/tpch/

APPENDIX. STATISTICAL RESULTS

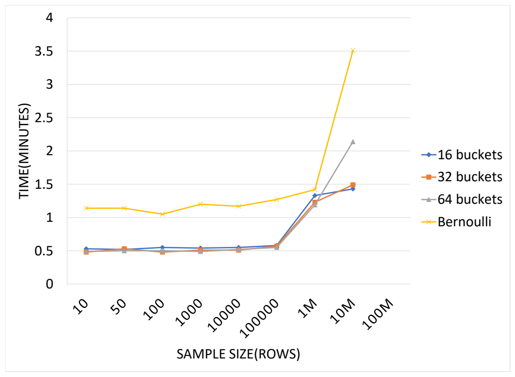


Figure A1 Comparison of time of processing and different samples sizes a different bucket count tables with 20M records.

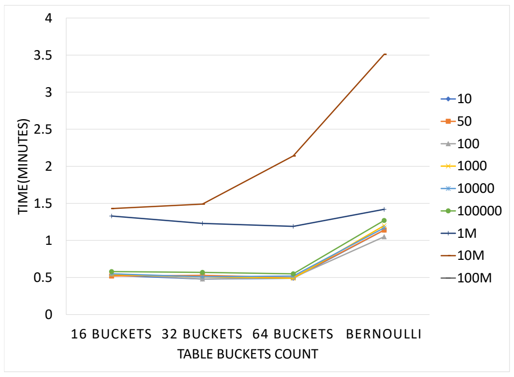


Figure A2 Comparison of time of processing and table with different bucket sizes of table with 20M records for different sample sizes.

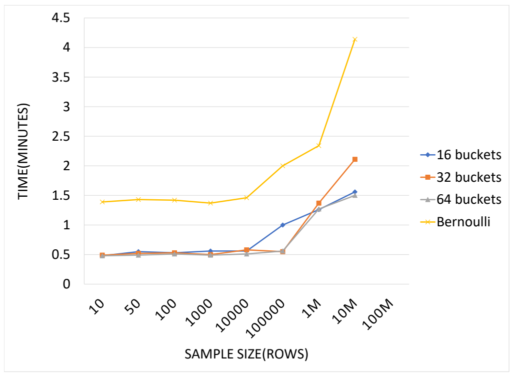


Figure A3 Comparison of time of processing and different samples sizes for a different bucket count tables with 40M records.

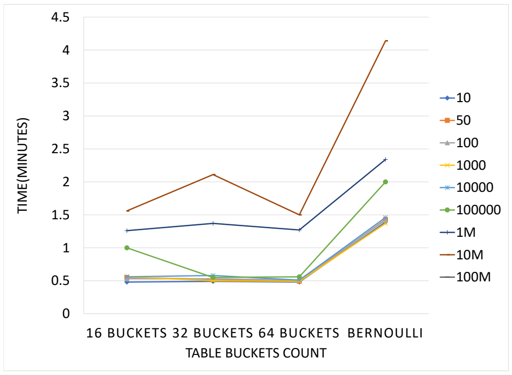


Figure A4 Comparison of time of processing and table with different bucket sizes of table with 40M records for different sample sizes.