Faster Query Response for Streaming Data using Probabilistic Data Storage Model

Ramesh Balasubramaniam, K. Nandhini

Abstract: ***Instead of focusing on making programmable changes for faster query response time our focus is on how the data is stored. In Streaming data, an efficient way to speed up delivery of results is by storing data compactly using a compresseddata storage model that will be beneficial in providing real-time analytics. In this paper, a data storage model named PDSM is presented, using Probabilistic data structures which provides answers to real-time queries on data streams. PDSM is a Probabilistic Data Storage Model that stores streaming data using hash functions as the primary data mapping method for incoming stream elements.Hash functions map a data set of arbitrary size to a data set of a fixed size. So regardless of source dataset size, the PDSM has a smaller fixed storage size which proves beneficial as a storage model.PDSM uses sketches (Count-min probabilistic data structure)to record the frequency of streaming data element occurrences and filtering (Bloom Filter probabilistic data structure)to check membership of elements in a stream. PDSM is usedto answer most sought-after queries, this way reducing query time on the most frequent questions.As PDSM uses Probabilistic data structures the accuracy of output results is an estimate and not 100% accurate. However, in streaming data, this is not a critical factor asall data elements in the stream are not required data, and a probabilistic estimate is a close enough answer to the query. Since storage used is minimal, this paper provides PDSMas a positive solution to faster query response time for streaming data and storage ofbig static datasets.***

Index Terms: Faster Query Response, Hashing, Little Memory, Probabilistic data Structures, Sketching.

I. INTRODUCTION

Given the magnitude of new sources of data in various sectors, there is a vast opportunity and competitive advantage for organizations to utilize this enormous (Big) data. Harnessing the power of streaming data from log files, sensors, social media, and other input sources businesses can be empowered to make decisions swiftly and confidently.

In data warehousing, 'Aggregate' is a summary table used to produce faster query response time on large data sets. Using a 'Group by' SQL query, a table is derived on large data sets. This paper is based on the words of Ralph Kimball, considered as one of the original architects of data warehousing [1]: "The single most dramatic way to affect performance in a large data warehouse is to provide a proper set of aggregate (summary) records that coexist with the primary base records. Aggregates can have a very significant effect on performance, in some cases speeding queries by a factor of one hundred or even one thousand. No other means exist to harvest such spectacular gains."

Revised Manuscript Received on May 10, 2019

Ramesh Balasubramaniam, Research Scholar,PG and Research Department of Computer Science, Chikkanna Govt. Arts College (Bharathiyar University), Tirupur, India.

Dr. K. Nandhini, Assistant Professor,PG and Research Department of Computer Science, Chikkanna Govt. Arts College (Bharathiyar University), Tirupur, India.

Aggregation methods are needed for streaming data also, but if the same data warehouse (database aggregate) concepts are applied it will result in high-latency processes, a massive amount of memory usage and a significant server configuration to address real-time use cases. So, applying Kimball's principal/concepts of aggregation, a different technique is needed to address new/current streaming datasets. All data analytics is not from historical data; both Stored historical data (data at rest) and incoming live streaming data (data in motion) are used to provide answers to queries. Database aggregation is used in Relational Database Management Systems (RDBMS) for query performance improvement. However, in today's big data/streaming models a parallel processing framework like Hadoop MapReduce technique and its ecosystem lead to large memory in the name node and a huge number of data nodes. In response to this problem, our study proposes on providing faster query performance with little process power, time & memory, with the one small drawback of having to give up minimal (negligible) accuracy percent. In streaming data, a probabilistic estimate is considered accurate enough during data analysis. PDSM is a Probabilistic Data Storage model that stores streaming data using Count-Min Sketch, and Bloom Filter data structures to answer most sought-after queries. Most queries are based on simple additive metrics like the total number of items sold, most frequently read news article, etc. This can very efficiently be summarized daily using PDSM. Count-Min Sketch records the frequency of streaming data element occurrences, and Bloom Filter does a membership check of elements in a stream. These data structures use hash functions as primary data mapping methods for incoming stream data elements. PDSM can be used as a data accumulator during query processing or a compact storage of raw data in stream-based computing.

II. BACKGROUND

Currently,the most widely used methods to improve query performance of large dataset are DatabaseSummary Table, In-Memory Database, and MapReduce Framework.

A. In-Memory Databases (IMDBs)

IMDBs rely on main memory to store data compared to the summary table method uses disk for data storage. As the required data is stored 'in memory', IMDBs have faster access time during processing and queries and are more predictable than disk mechanisms.

The cons are that they do need huge memory and bigger server requirements. According to Najajreh, et al. [2], in-memory databases (IMDB) though enhance performances thousands of times better than that of traditional hard disk systems, need special indexing techniques, recovery, concurrency controls and access methods to host the complete data in computer memory.

- Pros:
  - Faster query result
  - Accurate result
- Cons:
  - Need extensive memory
  - Need bigger server
  - Requires bigger software installation

B. MapReduce Framework

A MapReduce program is used for processing significant data streams using filtering and sorting techniques (Map procedure) and summary technique (Reduce procedure). By running parallel tasks on multiple nodes and managing data communications amongst the various parts of the system, MapReduce can reduce redundancy and provide fault tolerance. [3]

- Pros:
  - Query performance improves
  - Accurate result
- Cons:
  - Need large memory in name node
  - Need a very large number of data nodes
  - Requires bigger software installation

In the above methods, query performance is addressed but other factors like the need for a bigger server, longer processing time, and large memory is lacking, so a method which gives faster processing and query time, with minimal hardware configuration is needed.

III. METHODOLOGY

Our objective is to provide better data aggregation methods/techniques to address streaming data storage and query performance. The success of our proposed aggregate methodology is to give the following new capabilities to future streaming models:

- efficiently work with streaming data
- use very little memory
- use minimal data aggregation time
- quicker query response time
- allow reasonable degradation of accuracy or false positivity

A. PDSModel

In today's world given the vast volume, speed and diverse sources/formats of incoming data streams real-time applications need to have performance requirements met. Our method Probabilistic Data Storage model (PDSM) works as highlighted in figure (1).

First, input from various sources like DBMS, files, IoT, and weblogs are stored in the transaction table. The purpose of storing in a base (transaction) table is to retain historical data for analytical or other processing purposes. Next, using the transaction table, any number of probabilistic data storage (PDS)objects can be created. Depending on the most sought-after queries PDS objectsare created. After thecreation of the PDS object, real-time streaming data is passed to the transaction table and all current PDS objects. Each PDS is a summary of data for a specific query. During query retrieval, depending on the query the single PDS object or multiple PDSobjects (merged to provide result) is accessed.

For example:

- Top *n* trending news articles (Active News Articles PDS Object)
- How much revenue did the company make from consumer electronics or books or furniture? (Product group PDS Object)
- Top search keywords on a search engine (Search Word PDS Object)

Probabilistic Data Storagealgorithms use hashing techniques in Count-min sketch (CMS) that provides frequency count and in bloom filter thatchecks for membership. Below section describes hashing, CMS and Bloom Filter used in PDSM.

Hashing

For streaming data, hashing is used as the principle technique in many algorithms. It is used to speed up sorting, searching, inserting, or deleting data. It maps data of arbitrary size to data of fixed size. A hash table is a collection of items stored in such a way that it makes searching and finding easier. It reduces search runtime to O (1) when compared to a linear search runtime of O(n) and binary search runtime of O (log n). Hash functions have the following properties [4]:

- it always returns a number for an object
- two equal objects will always have the same number
- two unequal objects do not always have different numbers

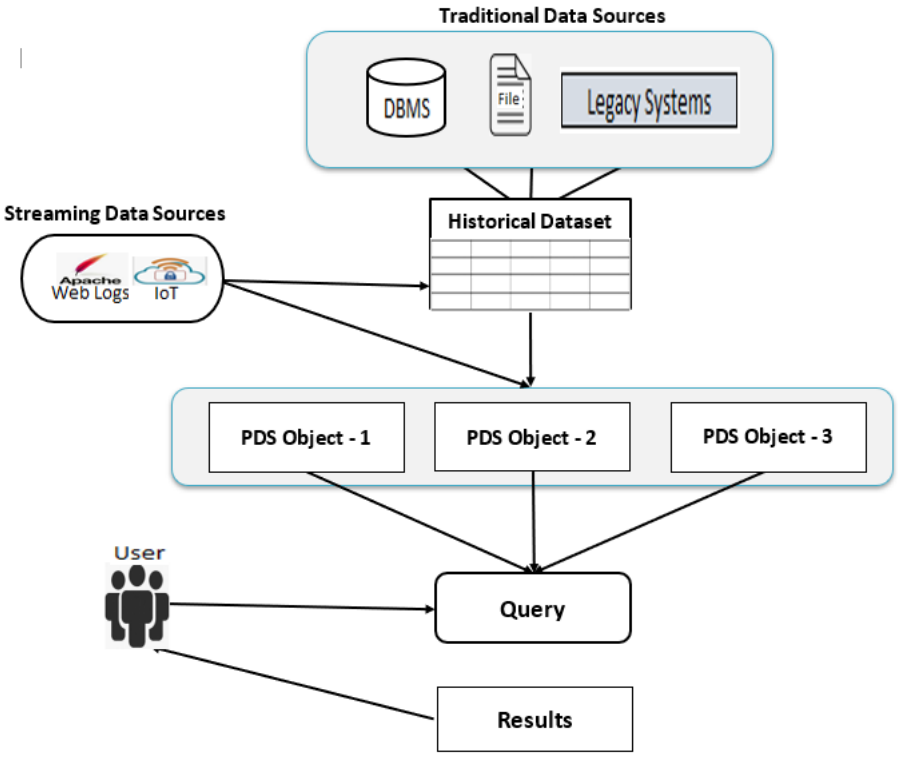

Fig 1. Probabilistic Data Storage Methodology

Frequency Count

Many algorithms address the frequency count problem. Starting from the Misra-Gries summary to find approximate frequent itemsets to Count-min sketch. Frequency count is all items whose frequency exceeds a specified fraction of the total number of items. Tracking approximate counts for a large number of objects arises in many scenarios such as most popular queries in a search engine, commonly purchased items by customers, frequently visited websites, most accessed topics in news streams or social media and such [5].

The algorithm of choice is the Count-min sketch (CMS) probabilistic data structure due to the benefits and findings that have been proved [6]:

- Regardless of dataset size once the accuracy threshold is reached it remains almost the same (unless the width size is too small)
- Even if the array width is changed once the accuracy threshold is reached beyond that point, the accuracy percent wavers down and up
- Again, regarding runtime, there is only a slight increase/decrease or no increase in the processing times when the width is increased
- Irrespective of increase in dataset size memory usage is also shown to remain constant

Membership Check

Filtering technique allows us to filter streams, so elements that belong to a particular set are allowed through, while most non-members are deleted. A large bit array and several hash functions are used. Members of the selected set are hashed to buckets, which are bits in the array, and those bits are set to 1. To test a stream element for membership, the element is hashed to a set of bits using each of the hash functions and only accept the element if all these bits are 1 [7].

Bloom Filter is a probabilistic data structure which saves memory space and is time efficient, but the trade-off is false positives. It tells us if the value is definitely not in the input stream or maybe in the stream. Bloom Filters do not allow false negatives, that is, returning false when the element is not in the set. To determine whether a value is not present in a search takes a longer time compared to determine whether the value exists. Bloom Filter reduces some failed searches to O(1) compared to O(log n) in a binary search or O(n) in a sequential search. So even though Bloom Filters allow false positives the simplicity, space saving and speed factors often compensate this downside. [8]

When a query comes in as a membership check, it will be done by a Bloom Filter and then if 'true,' meaning positive membership check, will proceed to further processing else the process stops and execution and processing time is saved.

B. Experimentation

In this section, the workings of PDSM and its features are summarized. Even though the PDSM works for any domain dataset,the experiment uses European online retail sales dataset and record the metrics. This dataset has volume, efficiently simulates streaming data and is also readily available for academic purposes. Our end goal is not to limit to one dataset but to provide our proposal of PDSM to various sectors and various streaming big data models.

Technologies used for this experiment are:

- Language - Python
- Library - Matplotlib

The transactional dataset [9] has 500K online retail sales data from 38 countries. The dataset contains two years (2010 and 2011) of data on 3826 SKUs for 4371 individual customers in the form of a CSV file. Each record has Invoice No, Invoice Date, Customer No, Country, SKU, Description, Qty, Unit Price and Amount.

A PDS Object is created based on the query requirement. For the experiment, 3 PDS Objects have been created - CNTY_SUM, SKU_SUM, and PROD_AVAIL but many can be created. The following queries will be addressed:

1. Last year sales quantity by country (frequency count)
2. Top10 products sold (heavy hitters)
3. Availability of Product in Inventory (membership check)

A skeletal Pseudocode (code 1) shows the creation of a PDS Object. Initially, all historical data is added into the PDS Object, and then Streaming data is added to the PDS Object as well as the base table (for historical purposes) in parallel as data arrives.

Code 1: Create Country Summary PDSObject

```
PDSM INIT (w,d)
for i <- 1 to num_ele do
(key, value) <= Incoming Streaming Data
 Add CNTY_SUM (key, value)
 Save CNTY_SUM
 Wait for next Streaming Data
repeat
Close PDS Object, Input Stream
```

Our first query is "Last year sales by country" CNTY_SUM PDS Object has summation value for each country and each year, the PDS Objectis queried, and the minimum value for each country is obtained. A skeletal Pseudocode (Code 2) shows the query of the PDS Object. The PDS Object returns the count value for each country.

Code 2: Query Country Summary PDSObject

```
Load CNTY_SUM
for i <- 1 to num_cnty do
    READ Member Item
    Value = PDS Object Query (member)
Output the result
repeat
Close Result file
```

The second query is "Top 10 products sold" which provides results for the top 10 products sold in a day. For this query, SKU_SUM heavy hitter PDS Object is created, that stores the top 10 product qty items in a list using Count-Min Sketch data structure. To find out the top items any time using the SKU_SUM query the list of heavy hitters is obtained which will produce the member and the value, which can change based on the input stream.

The third query is "Availability of Product in Inventory". Here PROD_AVAIL PDS Object is used to check for membership. The PROD_AVAIL PDS Object checks whether the Product SKU is in the filter. This PDS Object uses the pros of the Bloom filter data structure to provide results quickly. The optimal size for the PROD_AVAIL PDS Object uses the below two formulas [10]

Size of Bit array:

If the expected number of elements ***n*** is known and desired false positive probability is ***p*** then the size of bit array ***m*** can be calculated as:

$$m = -(n \ln P)/(\ln 2)^2 \qquad (1)$$

Optimum Number Of hash functions:

The number of hash functions ***k*** must be a positive integer. If ***m*** is the size of bit array and ***n*** is the number of elements to be inserted, then ***k*** can be calculated as:

$$k = m/n \ \ln 2 \qquad (2)$$

The result value can then be used for getting additional details like the inventory count of the available product also. A 'true' PROD_AVAIL PDS Object value sends the query to run against INV_SUM PDS Object (has the summary of product count in inventory) and the result is provided.

IV. RESULT ANALYSIS

A. Processing and Query time

For various dataset size 1K to 10K the processing and query time is found (Fig 2). Streaming data aggregation time which is the sketch creation time takes 0.03125 to 0.453125 seconds, and Query takes 0.015625 to 0.09375 seconds. Both sketch creation and sketch query time show a linear progression. Sketch creation is performed just once. The query result is retrieved very frequently; hence a lower query time shown in the experimentation is a positive response.

B. Memory usage

Regardless of dataset size, the memory usage remains fixed in a PDS Object. The SKU_SUM PDS Object uses 235 kb memory, and CNTY_SUM PDS Object uses 2 kb memory. Both PDS Object memory usage is based on the sketch size. Since the sketch size is fixed to the optimal width and number of hash functions the memory usage is minimal/negligible compared to the input data stream. This way the query and processing time is quicker. The PROD_AVAIL PDS Object uses 8 kb memory. Since PROD_AVAIL PDS Object is also based on the optimum number of bit array size and hash function count using the formula "(1), (2)," the memory remains constant irrespective of the increase in dataset size. Our experimentation highlights the small/tiny memory usage of PDSM.

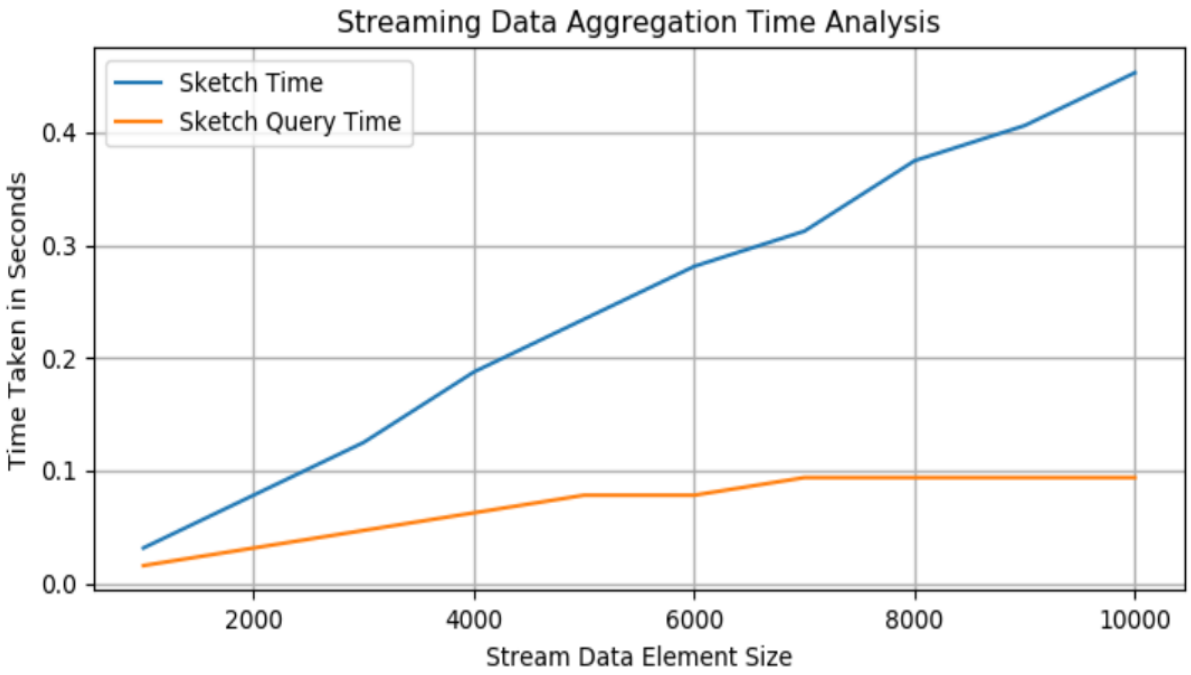

Fig: 2 PDSM Creation Time and Query Time

C. Accuracy

When the optimal width and depth (hash function count) is provided, the error rate can be reduced, and the accuracy of the PDS Object can be increased up to 99.99%. During the creation of the SKU_SUM and CNTY_SUM PDS Object, the confidence and error rate are passed as parameters which give accuracy percent to the user required levels. As stated, before PDSM gives a probabilistic count whereas other methods - Summary table, IMDB and MapReduce provide 100% accuracy. However, this is not critical in streaming data since all data are not required data and a probabilistic estimate is a close enough answer to the query.

V. CONCLUSION

The Probabilistic Data Storage model (PDSM) is an aggregate summary table that stores data based on probabilistic data structures to answer most sought-after queries quickly. PDSM sets out to provide answers to specific problems versus generic challenges. Depending on the query the Probabilistic data storage Object is created to provide faster query performance with less processing power& time andlittle/tiny memory. This has been extensively proved in the experimentation section. There are several ways in which query retrieval can be enhanced by increasing hardware through memory or processing power or by optimizing queries. Hardware improvements end up with bulky and eventually costly upkeep of servers. Query optimization has been a focus in many papers and queries are running at maximum efficiency. So, we have focused on how the data is stored in memory versus focusing on making programmable changes for faster query response time. PDSM is expected to provide valuable real-time support to data analysts during the time of decision making. This research work can extend to various sectors and various streaming big data models.

REFERENCES

1. R. Kimball, *DBMS - August 1996 - Aggregate Navigation With (Almost) No Metadata*, 12-Aug-1996. [Online]. Available: https://web.archive.org/web/20101211114831/http://www.rkimball.com/html/articles_search/articles1996/9608d54.html
2. Najajreh, Jihad, and Faisal Khamayseh. "Contemporary Improvements of In-Memory Databases: A Survey." *2017 8th International Conference on Information Technology (ICIT)*, 2017, doi:10.1109/icitech.2017.8080059.
3. J. Dean and S. Ghemawat, "MapReduce," *Communications of the ACM*, vol. 53, no. 1, p. 72, 2010.
4. B. N. Miller and D. L. Ranum, *Problem solving with algorithms and data structures using Python*. Decorah, IA: Brad Miller, David Ranum, 2014
5. G. Cormode, "Misra-Gries Summaries," *Encyclopedia of Algorithms*, pp. 1–5, 2014.
6. R. Balasubramaniam and K. Nandhini "Efficient Count-Min Sketch to solve Frequent Items problem" *Proceedings of the First International Conference on Electrical and Computer Technologies 2019 (ICAECT 2019)* "to be published. "
7. B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
8. Habimana, Jean. "Query Optimization Techniques - Tips for Writing Efficient and Faster SQL Queries." *International Journal of Scientific & Technology Research*, vol. 4, no. 10, Oct. 2015, pp. 22–26.
9. J. S. Seo, "Online Retail Data Set from UCI ML repo," *Kaggle*, 22-Jan-2018. [Online]. Available: https://www.kaggle.com/jihyeseo/online-retail-data-set-from-uci-ml-repo.
10. A. Kumar, "Bloom Filters - Introduction and Python Implementation," *GeeksforGeeks*, 08-Feb-2018. 1Online]. Available: https://www.geeksforgeeks.org/bloom-filters-introduction-and-python-implementation/.