Increasing the Performance of Iceberg Query Through Summary Tables

Gohar Rahman[1], Wajid Ali[2], Mehmood Ahmed[3], Hassan Jamil Sayed[4], Mohammad A. Saleh[5*]
Faculty of Computing and Informatics, University Malaysia Sabah (UMS), 88400 Kota Kinabalu, Sabah Malaysia[1, 5]
School of Computer Science and Technology, Jilin University, Changchun, Republic of China, China[2]
Department of Information Technology, The University of Haripur, Haripur, KP, Pakistan[3]
Asia Pacific University of Technology & Innovation (APU) Bukit Jalil, Kuala Lumpur, Malaysia[4]

***Abstract*—One of the key challenging problems in data mining is data retrieval from large data repositories, as the sizes of data are growing very fast, to deal with this situation, there is a need for efficient data mining techniques. For efficient mining tasks number of queries have been emerged. Iceberg query is one of them, in which the output is much smaller like the tip of the iceberg as compared to the large input dataset, these queries take very long processing time and require a huge amount of main memory. However the processing devices have limited memories, so the efficient processing of iceberg queries is a challenging problem for most of the researchers. In this paper we present a novel technique, namely a summary table, to address this problem. Specifically, we adopt the summary table technique to acquire the required results at summary levels. The experimental results demonstrate that the summary table technique is highly effective for large datasets. Compared to bitmap indexing and cubed techniques, the summary table offers faster retrieval capabilities. Furthermore, the proposed technique achieved state-of-the-art performance.**

Keywords*—Threshold (TH); bitmap index; aggregate function; Iceberg Query (IB); anti-monotone; non-anti-monotone aggregation

I. Introduction

Data retrieval and storage play a very important role in databases. The effectiveness of data retrieving techniques depends on specific. Since few years many queries have emerged, one of them is the iceberg query (IBQ) in which the output is significantly small as compared to the input, such query is called IBQ, where the number of above-threshold outcome is usually very small like the tip of an iceberg as compared to large amount of input data [1]. This is a unique class of aggregation queries connecting HAVING () and GROUP BY () clauses, which computes aggregated values below or above a given threshold (TH). This query is first introduced in data mining (DM) [2]. Most of the data DM queries are IB queries. Several applications use aggregate functions such as, Min (), Avg (), Max (), Sum (), and Count () over an attribute or set of attributes to find aggregate values greater than a particular threshold, these aggregate values above the threshold values give more importance. The RDBMS, e.g., MySQL, Postgre SQL, SQL Server, Oracle, DB6, Sybase, and column-oriented databases e.g., Lucid DB, Vertica, and Monet DB all use common aggregation algorithms that first aggregate all rows and then calculate the Having () clause to select the iceberg result [3].

An iceberg query has the following characteristics: (a) Computing aggregate functions on one or more attributes (b) Dealing with large data sets, containing large unique attributes combination (domain size), and (c) Returning results below or above a given TH. These queries face some problems during executions, like 1) It needs to execute within a limited memory, which means memory size is lesser than domain size 2) Computation of aggregation values takes a large amount of time.

The global objective of this work is to reduce the execution time of the iceberg queries within a limited memory. Today's world is rich in data; every organization and social media generates and stores huge amounts of data which need an efficient way to deal with. For this purpose, IB query is an ideal choice. These queries are used in many applications. Including market basket analysis [4] means finding item pairs (or triplets etc.) that are bought together by many customers in large data warehouses. In other words, market basket analysis means a collection of items purchased by a customer in a single transaction. It is based on two key attributes considered for the threshold value used for finding item pairs; these attributes comprise support and confidence. If support and confidence values are above or below some specified threshold then it identifies products and their content that go well together. Similarly, clustering [5] is a process of partitioning a set of records into groups (clusters), such that all records in a group are related to each other and records that belong to two different groups are different [6]. This helps users to recognize the natural grouping or structure in a data set and this natural grouping is done in clustering based on some specific threshold values in each IB query [7].

A. *Properties of Aggregation Function*

Aggregation function is one the key part of iceberg queries, such as Sum (), Count (), Min (), Max (), and Avg (). Aggregate function is divided into two types (1) Anti-Monotone, and (2) Non-Anti-Monotone aggregation function [8]. An anti-monotone uses apriori [4] property, but non-anti-monotone are not able to use apriori property, examples of anti-monotone are Count (), Sum (), Max (), and Min (), whereas non-anti-monotone are Avg () and Div (). The main benefit of using IB with anti-monotone function is the pruning of computing aggregation functions reduces the time to produce the required query result [9]. On the other hand, non-anti-monotone aggregation IB queries don't take advantage of threshold on Avg () values as anti-monotone aggregation takes on Min (), Sum (), Max (), and Count (). Average IB queries compute average for all unique grouped attributes, and then apply threshold constraint on those average values [10]. To deal with aggregation functions there is a high gape between the researcher’s contribution toward these two types with a ratio of 22 to 78 percent [11] as shown in Fig 1.

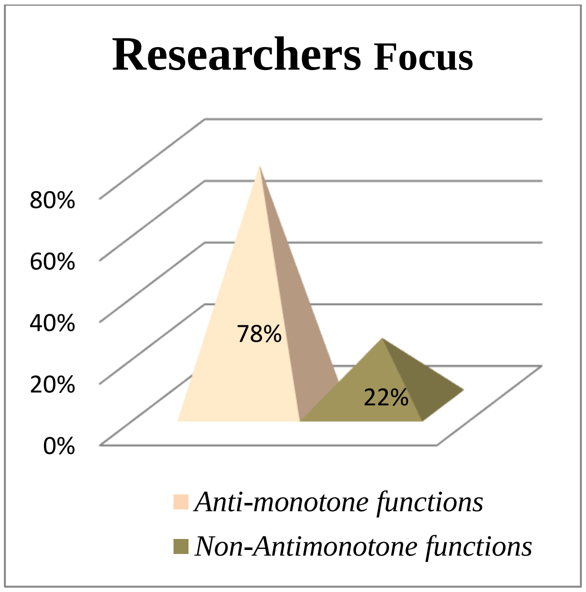

Fig. 1. Aggregation functions.

The rest of the paper is structured as follows, section II presents a review of related research. Section III describes the proposed technique, proposed architecture, and implementation. Section IV describes the results and analysis, and in Section V conclusion and future direction are presented.

II. Review of Related Research

Since a few decades iceberg query has always been an active area of research. Researchers have provided different guidelines and suggestions to improve its performance. We are going to discuss some works in literature based on different specific categories.

A. *Bitmap Index Techniques*

To accelerate the IB queries, bitmap indices are one of the well-organized and well-known choices in column stores and data warehousing applications. Spiegler et al. [12] first introduced the concept of bitmap index (BI). Basically, a matrix of 0 and 1 bit’s makes a bitmap. Its size depends on the number of matchless attributes that exist in vector upon which bitmap is created. Basically, a bitmap index is used to index values of a single column in a table. For illustration Table I indicates a bitmap index with nine rows, and column Y, where column Y is indexed with integer values from 0 to 3 and its cardinality becomes four because it has four different values. Columns X0, X1, X2 and X3 with subscripts signify bitmap index for Y contains four bitmaps. The second bit X1 in Table I is 1 because the second row of Y contains value 1, while corresponding bits of X0, X2 and X3 are all 0 Vuppuand Rao [13] has presented a new evaluation scheme for processing IB queries using bitmap index position. They developed an algorithm based on retrieving index positions of all 1's from each bitmap. Further, these indices positions are processed by using commonality condition which selects whether the pair of directions is iceberg result or not. To retain for future reference, an XOR operation is conducted for bitmaps, which is the iceberg query result. Their experiments show that algorithms which is based on index positions takes less processing time to answer IBQ.

TABLE I. Bitmaps Index for Column Named Y

RID	Y	X0	X1	X2	X3
0	2	0	0	1	0
1	1	0	1	0	0
2	3	0	0	0	1
3	0	1	0	0	0
4	3	0	0	0	1
5	1	0	1	0	0
6	0	1	0	0	0
7	0	1	0	0	0
8	2	0	0	1	0

Padmapriya and Shanmugapriya [1] introduced an index based IBQ assessment method. The key aim of using the index is to convert the bit value into an integer value which speeds up the query evaluation process and takes less memory. This technique performed well on the state-of-the-techniques. O'Neil [14] Model 204 was the first model used bitmap index for wide-spread commercial product making. This was a combination of row identifiers (RID list) and basic bitmap index without compression. In general B+ tree index technique is like the performance of Model 204. Prakash et al. [15] presented a bitmap index as a better choice for querying huge and multidimensional scientific datasets. They have developed a well-organized algorithm based on retrieving index positions of all 1’s from each bitmap. These index positions are further processed on common features which decide whether the pair of vectors is IB result or not. Generation of the decision algorithm which involves pre-processing of data sets through bitmap indexing approach is the global objective of [16]. The key benefits of this index strategy are load balancing, identifying frequent patterns of the data sets, kind of data types available in the databases, slowly changing dimension scenarios handling and usage of aggregation in the form of IB querying.

Shankar et al. [17] introduced a cache-based evaluation technique for IBQ by taking threshold value equal to 1 using a compressed bitmap index, and for future situation the required results are saved in cache memory. In future it just picks up the required results from the cache memory as a substitute of executing once again on the database table. Therefore, this approach clearly states that, an execution time of IB query is improved by avoiding duplication of evaluation process several times. In this work testing was conducted by applying an IB query stated on the database table which consists of one million rows with two attributes X and Y, by using COUNT () aggregation function. IB query evaluation method was the first function applied to accept all those tuples as an input and produces the iceberg results with its count value fulfilled by threshold greater than or equal to 1. Then these results are given as an input to the second unit catching IB results in an ascending order. For future position this unit saves the iceberg results in cache memory. The last unit which takes results from cache unit is responsible to answer an IB query for thresholds greater than 1, is just selected from the cache memory and send to output. This cache-based technique enhanced the overall processing time of IBQ for an efficient data retrieval task.

Laxmaiah, Govardhan and Kumar [18] have presented an efficient Database Priority Queue (DPQ) algorithm for processing IBQ using compressed bitmap Index; the impact of this method was to speed up the query evolution method by emptying the compression queue. In this technique, first the iceberg query is responsible to select the similar words with aggregate attributes Y and X from the relation R in which the TH value is taking between 1000 and 9000. By taking a database table which has two attributes Y and X which contain millions of rows and using count () aggregate function. Then the experiment is conducted by applying an IBQ on the first function that generates bitmaps accepts all those rows as an input. The key achievement of this technique is keeping the comparable number of rows in a relational database table and at the same time keeps the results with density queues; consequently, further this experiment is repetitive for different iceberg threshold as well.

Zianget al. [19] suggested a well-organized algorithm for IBQ processing by using compact bitmap indices. The given algorithm does not depend on any testing compression process and demonstrates better performance over presented schemes. Bitmap index has three attractive benefits based on observation such as: first, conducting bitwise operations that reduce computation time. Second, saving disk space by avoiding rows scan on a relation by using attributes group. Third, by leveraging the anti-monotone property of IB therefore this algorithm is not affected by the number of diverse values, and length of attributes in the relation. Rao et al. [20] presented a well-organized technique, known as dynamic pruning technique or vector alignment algorithm to answer IBQ by using compressed bitmap indices, this algorithm guarantees that no empty result is generate by using any bitwise-AND operation. Bitmap indices are presented to get more improved results as compared to tree based index method such as alternatives of R-tree or B-tree [21], explicitly, this work is motivated to compute IBQ using bitmap indices as an index pruning based approach.

Otoo and Shoshani [22] introduced bitmap indexing pattern algorithms for little cardinality attributes to analyze the time and space complexities of Byte Aligned Code (BBC) and Word Aligned Hybrid (WAH) compressed bitmap indices. To demonstrate their success for using high cardinality attributes, for high cardinality attributes, c << N, here c represent compressed bitmap index and N represent the number of words, the WAH algorithm compressed indices uses define 2N words, this 2N words is about half the size of a representative B-tree index. On the other side BBC compressed indices are even smaller but it also represents an in-place algorithm that is linear to the total size of the bitmaps involved to OR many bitmaps in time complexity. The whole size of the bitmaps used is proportional to the number of hits in the worst-case situation. By using compressed bitmap, it shows to search one attribute is optimal and this optimality is established with timing results from a set of real application and random data. In these sets of examinations, WAH compressed bitmap indices were nearly twice as fast as BBC code compressed indices. Both indices could achieve search operations faster than the projection index by using worst cases, on average. By using the WAH compressed indices, time is not more than the projection index. Bitmap indices in study [14] using bitmap vectors for vertical organization of a columns. Every vector characterized the presence of a distinctive value in the column across all rows in the table.

In study [23] an effective bitmap pruning strategy was introduced, which is grounded in order of high cardinality in Priority Queue (PQ) by using compressed bitmap indices for processing an IBQ. By using this method, it allowed the movement of vectors to enter PQs on the high count 1' to get additional benefit for large pruning of bitmaps. The pruned vector essentially improved the response time. Processing huge quantities of data in predetermined time factor is a key challenge faced by data warehousing. By using [24] bitmap indexing which is extra meaningful in quicker data processing generated a strategic decision technique for data warehousing environment. For managing 'Boolean' kind of data, like gender, the bitmap indexing is best suited, such as false and true group of values. Bitmap indexing mainly depends on 1's and 0's kind of data. The data is openly processed by using CPU, which does not support any alteration of the data items into a new format, and greatly decreases the processing time of the records. This work shows the integration of IB querying; within the identified amount of time factors while processing huge amounts of data in data warehousing environments and achieved efficient results. Most of the previous research work mainly centered about identifying "well behaved "constraints with respect to constraint pushing [25, 26], this work proposed a novel pushing technique known Divid-and-Approximate (DnA), which combine two ideas, "Approximate Push" and "Divide-and-Conquer" to generate a strongest constraint for pruning with non-anti-monotone aggregation constraints in IB cubing. The key idea of DnA was to divide a partition of tuples into two subspaces of positive and negative degree values, so that a given constraint could be rewritten using monotone or ant monotone constraints in subspaces. These works mainly focused on (a) SQL like tuple-based aggregates, rather than item-based aggregates (b) General aggregate constraints, rather than only "well behaved", and (c) Constraint independent methods, rather than constraint specific methods. The idea of DnA contributed a new share to constrain data mining techniques.

Laxmaiah et al. [27] presented an efficient Density Priority Queue (DPQ) procedure for an IBQ by using compacted bitmap index based on two stages (1) Using an algorithm for pruning the vectors dynamically by computing newest counts for reinsertion and certifies the proposal using a sample database. By dropping the bitmap vectors dynamically using a high-count attribute to calculate an IB query, and (2) Using a validation of DPQ approach on RDBMS section to show the validity of the proposed DPQ and evaluates an IB query having COUNT () aggregate function. As compared to previous strategies PQ is a more sophisticated technique. Based on large data sets the experimental results indicate significant progress which proves the effect of IBQ computation.

In study [28], the distributed Iceberg Semi-Join operator is proposed, which is used in most of the real-life applications.

This technique is used to get information from two different independent data marts or from a remote digital library and use an efficient technique, which insert the execution of the IBQ and join in the two servers by using Mul-FIS; to prune the non-qualifying groups it uses. This work provides important advantages over its competitors. By using multidimensional databases each dimension is nothing, but one subject oriented table with attributes of related metrics [29], writing queries on multidimensional databases are difficult and include join operations due to which the reply time of query is increased on a massive database. By using queries with aggregation function, and summarization is followed by using 'having' clause. This type of query is very complex and requires extra time. This work focused on two different bitmap indexes search implementations techniques, such as RIDB and Fast Bit. The key improvements in Fast Bit are the Word-Aligned Hybrid (WAH) compression for bitmaps and multilevel bitmap encoding approaches. Fast Bit index is typically greater than RID Bit index, in fewer intervals of time it can answer several queries, as it accesses the required bitmaps in fewer I/O operations [30]. RID Bit normally costs less CPU time in answering queries than that of Fast Bit, though, the CPU time differences are minor matched with I/O time. A brief comparison between "Fast Bit" and "RID Bit" is discussed in [31]. At the end this section Table II categorizes some basic characteristics of bitmap Indexing.

B. *Based on Compound or Hybrid Algorithms*

In study [31], four algorithms namely Partitioned Tree (PT), Breadth-first writing Partitioned Parallel BUC (BPP), Replicated Parallel BUC (RP), and Affinity Skip List (ASL) are introduced, these algorithms are calculated experimentally over a range of parameters to get the necessary condition in which the algorithms could outperform. The Key features of the proposed algorithms are mentioned in Table 2, which described all four algorithms with respect to their writing strategy, Data decomposition, Load Balance, and Relationship of cuboids.

Matias and Segaly [32] presented two effective algorithms based on hash partitioning technique to compute estimated IB queries. Using a hash function to divide a data set into specific values that was independent from a subset resulting with properly smaller independent sub problems that can be handled efficiently with certain performance. In [33] two algorithms which use a concise sample and basic component have been presented. The first algorithm is used to sort the sequence into the necessary number of partitions and the second algorithm is used for computation. Though acting only one pass over the sequence these algorithms are used to compute the approximation query, without accessing a database and without materializing data sets which are stated implicitly, therefore it can be applied online for streaming data. In [34] the author has emphasized two problems; (1) Efficiently classify passing stories from rapid streaming social content and (2) To execute IB queries to form the structural background between stories. To give attention to the solution of the first problem, the social stream is converted into a time gap of tube network, and model passing stories as (k, d) cores in the tube network.

Two polynomial time algorithms were proposed to extract maximal (k, d) cores. The second problem, deterministic context searches and randomized context search is applied to maintain the IBQ efficiently and carefully, which permits performing context search without pair wise relationship.

TABLE II. KEY FEATURES OF FOUR ALGORITHMS

Algorithms	Writing Strategy	Data Decomposition	Load Balance	Relationship of cuboids
RP	Depth-First	Replicated	Weak	Bottom-up
BPP	Breadth-First	Partitioned	Weak	Bottom-up
ASL	Breadth-First	Replicated	Strong	Top-down
PT	Breadth-First	Replicated	Strong	Hybrid

By spreading the probabilistic techniques and suggested hybrid and multi buckets algorithms for processing of IB queries was first considered by study [35]. The sample and multiple hash function are used as an important building chunk of probabilistic events such as scaled-sampling course and count algorithms. It projected the sizes of a query results in order to expect the valid IB results, which decreases memory requirements and raises aggregate query performance. Though, these techniques incorrectly resulted in false negatives and positives. To overcome these bugs, an efficient approach is planned by hybridizing the sampling and coarse count techniques, such as hashing technique that allocated a bitmap of size 'M' in the memory is constructed on linear counting algorithm (LCA). In this method, all entries are initialized with '0's. The linear counting algorithm applies a column interest and then scans the relation. On the other hand, the hash function produces a bitmap address, and the algorithm sets this addressed bit to '1'. This algorithm first counts the number of empty bitmap entries. Then it guesses the column cardinality by distributing the count by the bitmap size 'm' and plugging the given result which increases the overall performance of hybrid technique.

III. PROPOSED TECHNIQUE

In the previous section we discussed in detail IB query processing techniques and algorithms. Researchers have introduced different algorithms and techniques for increasing the performance of iceberg query. Some of the existing techniques focus on the SUM (), MIN (), MAX (), AVERAGE (), and COUNT () aggregate functions, such as, bitmap indexing techniques, cubed techniques, AND operation techniques, POP operation techniques, attribute-based techniques, and hybrid algorithm techniques. All these techniques have some limitations, such as, some techniques have the deficiency to occupy more space in memory, some techniques slow down the system performance, some require complex algorithms which are difficult to maintain, and some take more time to produce the required result in a required time. To overcome the limitations of existing techniques to improve the performance of IBQ, an enhanced technique based on, summary table is proposed to improve the IB query performance. This technique improves the running time of the query for searching a specific record and reduces the elapse time. This section aims to discuss the details of the proposed work using summary table's technique for IB query processing to finds out the required result greater than the given TH.

The proposed technique is applied on sample customer tables of different sizes with the same attributes, such as customer identity (Cus_Id), Expense per day (Cus_Exp-per-day), and job (Job). The values of Cus_Id, Cus_Exp-per-day, and Job attributes classify each group, while Cus_Exp-per-day refers to the field on which the aggregate operator Count () is being computed based on a specific TH value. The focus of this work is Count () aggregate function which is applied on (Cus_Exp-per-day) field, the scope of the proposed work is to find expenses of all those customers whose expense is greater than some specified threshold values. The proposed work is better than the existing work in different perspectives. Different summary tables are created in the proposed work and IBQs are used to extract the required results from these summary tables by ignoring scanning whole data sets one by one. As compared to the proposed technique, the state-of-the-art technique was used to scan all the tables for the required data results which slowed down the query processing time. The advantage of our proposed technique is to improve the running time of the query for searching a specific record and reduces the elapse time.

A. *Proposed Architecure*

The proposed architecture is a robust and efficient summary table creation system based on different threshold values for identifying how summary tables are created from the original tables. Fig. 2 draws our novel architecture, which consists of three phases: an Execute1 phase for processing simple IB queries, that is directly applied on original source table for extracting the required dataset, it scan the whole table for the required data set based on a specific threshold value, the drawback of this technique was the requirement of huge amount of processing time which effect the performance efficiency of this technique, in contrast the Execute phase in the mentioned architecture was used to automatically creating summary tables from original table, and then on those summary tables the required query are executed based on a specific threshold value for the required result instead of scanning the whole table. In the proposed architecture only eight summary tables are mentioned for the sake of simplicity these summary tables are used as a source of IB queries, instead of scanning the whole original data set, and the third phase displays the required output of the processed IB query.

B. *Implemetation*

This section describes different techniques that were planned in the preceding section. The first step toward implementation was by taking different target customers tables ranging from 50K to 500k data set, which store customers' related records. In the second step the existing technique is applied to fetch customer's records by using twenty different threshold values ranging from 1k to 20k on each customer's table. Based on the mentioned threshold values, the times taken by the existing technique are recorded and then the average time is calculated for ten run cycle on a given table. The same experiment is repeated ten times for each data set in total. Similarly, the proposed technique is represented in the same way on the same data accordingly and twenty summary tables are created for each target table, that store pre-calculated aggregate results of the COUNT () aggregate function. The proposed technique then considers the threshold values given in the HAVING clause of the IB query and fetches results from the respective summary tables as per the given threshold values.

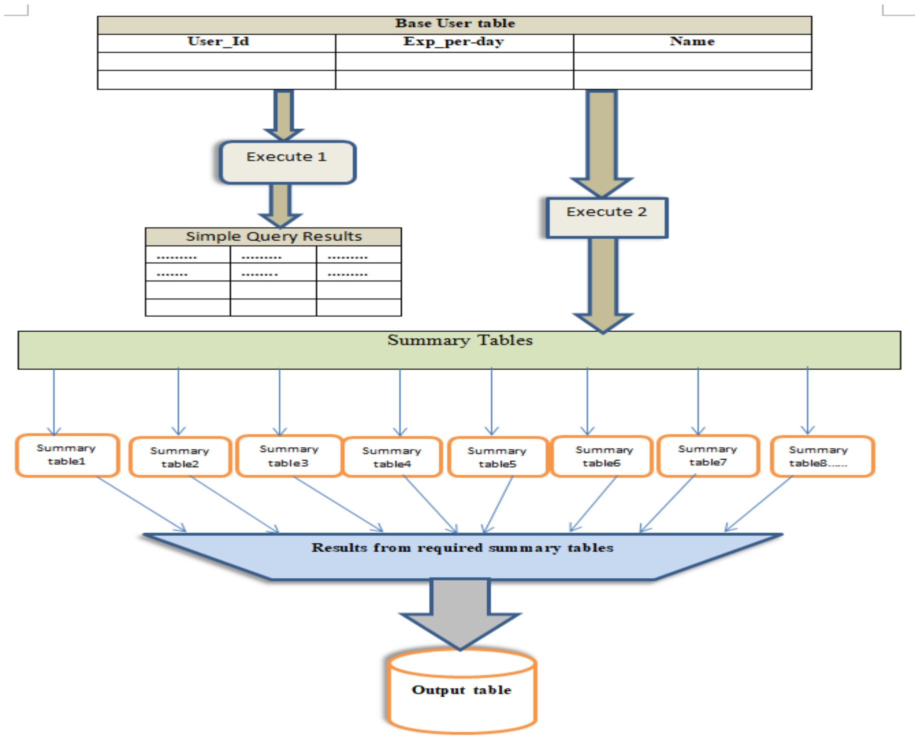

Fig. 2. Proposed architecture.

IV. Result and Analysis

In this section, results obtained during experiments are analyzed. Table III and Fig. 4 represent comparison among existing and proposed techniques based on query performance for different dataset ranging from 50k to 500k. The first column in Table III indicates different data sets, the second and third column correspond to average processing time of both techniques by using ten cycle of run count testing condition on each specific data set accordingly. The "Existing technique time" and "Proposed technique time" represent the average processing time of the existing and proposed IB query by retrieving the required result. The recorded values show high differences in execution times. e.g. when the dataset was 50k, then the corresponding average values of ten count cycle of existing and proposed techniques was about 6.2ms and 1.02ms, similarly for 100k the corresponding values is 6.8ms and 1.1ms, and the same is recorded for the other data sets till to 500k respectively. The above tabulated values are shown in the following Fig. 3 to understand well to the reader, the x-axis represents different datasets and y-axis represents execution time. Then how the execution times of existing and proposed techniques are gradually varying with different datasets.

A. *Comparison between Simple and Proposed Techniques*

In Table IV, we drawn the comparison between simple and proposed techniques only for one dataset of size 50k, there are nine others different tables is used to store the same comparison used in Table III with different data set (100k, 150k, 200k, 250k, 300k, 350k, 400k, 450k, and 500k) during the whole experiment to record the processing time of both techniques, which is not mentioned in this section due to a large number of comparison. Table IV represents the comparison of "Average of ten run cycle" among simple and proposed query processing. In the Table IV "RECORDS" field represent the number of records in given table, "Cycle of Run Count" represents ten counts of query processing of each proposed and simple technique, "Simple Query Technique" field represent the state-of-the-art technique, "Proposed Query Technique" field represents the proposed technique and "AVERAGE OF ALL" field represent the average time of ten cycle processing of each proposed and simple query technique. To comprehend well to readers, we draw the above tabulated data in graph form which are shown in the following Fig. 4. The x-axis represents the number of processing run count ranging from 1 to 10 cycles, and as well as average of all run count cycles for both simple and proposed techniques, and y-axis represents the execution time in microsecond.

TABLE III. Existing and Proposed Technique Time Interval

Dataset	Existing technique time	Proposed technique time
50K	6.2ms (average time of ten run cycle)	1.02 ms (average time of ten run cycle)
100K	6.8ms	1.1 ms
150K	8.7ms	1.3ms
200K	10.5 ms	1.6 ms
250K	12.5 ms	1.9 ms
300k	14.7 ms	2.1 ms
350K	16.5 ms	2.3 ms
400k	18.3 ms	2.7 ms
450k	20.1 ms	2.9 ms
500k	22.4 ms	3.1 ms

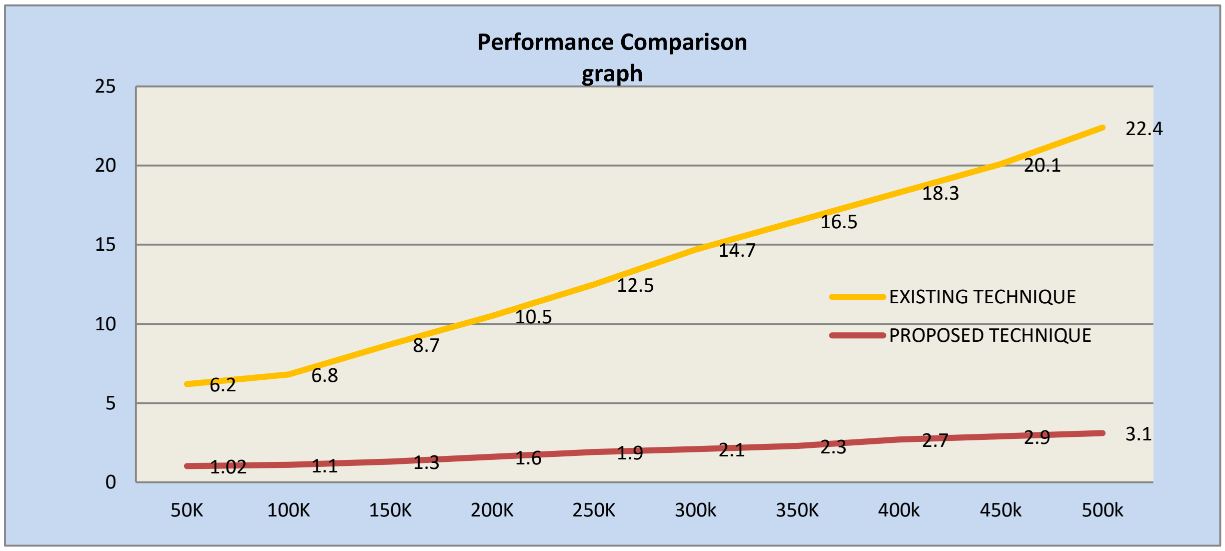

Fig. 3. Performance comparison of existing and proposed techniques.

TABLE IV. COMPARISON OF CYCLE OF RUN COUNT BETWEEN PROPOSED AND EXISTING TECHNIQUE

Table 4	50k Records										
Cycle of Run Count	1	2	3	4	5	6	7	8	9	10	AVERAGE OF ALL
Simple query processing time	6	6.01	6.05	6.07	6.08	6.1	6.3	6.6	6.7	6.9	6.28
Proposed query processing time	1	1.02	1.03	1.05	1.05	1.06	1.08	1.09	1.1	1.4	1.08

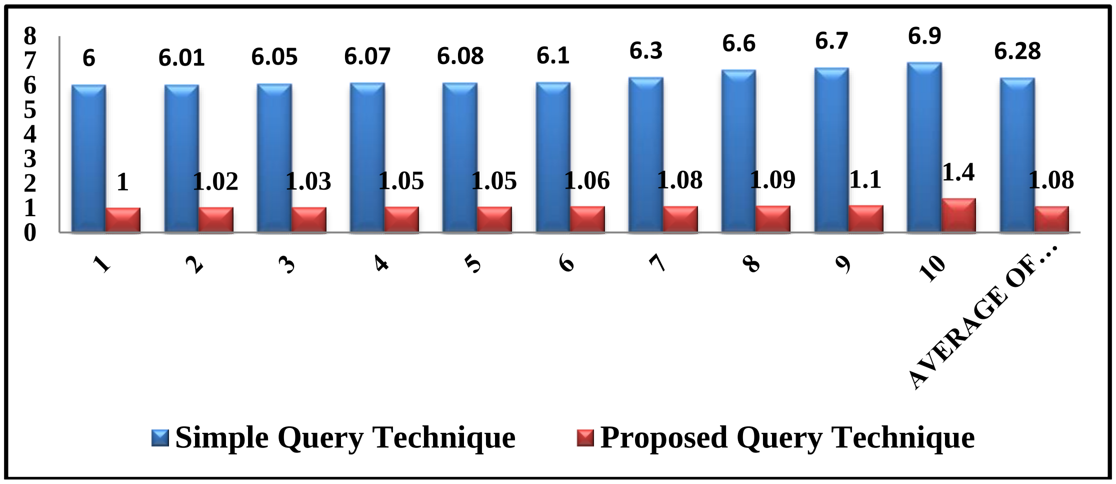

Fig. 4. Comparison of cycle of run Count between existing and proposed technique.

V. CONCLUSION

In this paper, we proposed a summary table technique for processing iceberg queries efficiently. According to IBM 2.5 quintillion bytes of data are generated by different electronic devices on a daily basis. Extraction of useful information from those huge data sets is a well-known challenging problem. From the last few decades most of works have focused to increase the performance of IB queries with respect to time constraint, computing memory, data repositories, computing memories, and data scanning. In this paper our technique is highly simple and different as compared to the previous works based on bitmap indexing and cubed technique. The summary table technique leverages the IB queries at the summary tables; these summary tables are created dynamically from the base table based on different threshold values ranging from 1k to 20k before the execution of queries. At the time of issuing the query, the proposed technique considers the threshold given in the query and fetches the calculated COUNT () aggregation from that specified summary tables as opposed to recalculating the aggregate function from the base table to produce the required results. Our method has improved the main metrics significantly. However, to increase the performance of IB queries in a large dataset is still a big challenge, in the future, this research work focused on the COUNT () aggregate function. In future we would like to extend our work to other aggregate functions such as MAX (), MIN (), SUM (), and AVG (). In this work we only considered the high-level IB queries. Similarly, we would like to continue working on low-level queries as well.

ACKNOWLEDGMENT

The authors would like to thank the University Malaysia Sabah for supporting this research Under the Grant NALS/ SLB2451NALS.

REFERENCES

[1] Ramírez-Gallego, S., García, S., Bergmeir, C., Triguero, I., Mendoza, C., & Herrera, F. (2018). Fast distributed big data preprocessing using Spark. IEEE Access, 6, 21216-21231

[2] Glavic, B., & Alonso, G. (2021). Verifying data-centric programs: Are databases the new bell-bottoms? Communications of the ACM, 64(7), 93-101.

[3] Godfrey, P., & Shipley, R. (2020). Iceberg queries revisited: A multi-dimensional perspective. Proceedings of the VLDB Endowment, 13(11), 2627-2639

[4] Müller, R., & Bach, F. (2023). Optimizing iceberg queries in modern database systems: A comprehensive survey. ACM Computing Surveys, 56(4), 1-36.

[5] Jinuk Bae and Sukho Lee. 2000. Partitioning algorithms for the computation of average iceberg queries. In Data Warehousing and Knowledge Discovery, Springer BerLin Heidelberg, pp. 276-286.

[6] Kevin Beyer and Raghu Ramakrishnan. 1999. Bottom-up computation of sparse and iceberg cube, In ACM SIGMOD Record, vol. 28, no.2, pp. 359-370.

[7] Raghu Ramakrishnan and Gehrke Johannes. 2000. Database management systems, McGraw-Hill New York, vol. 3, pp. 1-1104.

[8] WP Yan and P Larson. 1994. Data reduction through early grouping. In Proceedings of thconference of the Centre for Advanced Studies on Collaborative research, pp .1-74.

[9] Khaled AlSabti. 2006. Efficient Computing of Iceberg Queries Using Quantiling. Journal of King Saud University-Computer and Information Sciences, vol. 18, pp. 53-75.

[10] Ricardo Baeza-Yates.1992. Information retrieval: data structures & algorithms, Prentice Hall.

[11] Usama Fayyad, Gregory Piatetsky-Shapiro and Padhraic Smyth. 1996. From data mining to knowledge discovery in databases, AI magazine, vol. 17, no.3, pp. 1-37.

[12] Israel Spiegler and Rafi Maayan. 1985. Storage and retrieval considerations of binary databases, Information processing and management: an international journal, vol.21, no. 3, pp. 233-254.

[13] Vuppu Shankar, and CV Guru Rao. 2013. Computing iceberg queries efficiently using bitmap index positions. In Human Computer Interactions (ICHCI), IEEE International Conference on, pp. 1-6.

[14] PE O'Neil. 1989. Model 204 architecture and performance. In High Performance Transaction Systems. Springer Berlin Heidelberg, vol. 359, pp. 39-59.

[15] Prakash, Kale Sarika, and PM Joe Prathap. 2015. Bitmap Indexing a Suitable Approach for Data Warehouse Design. International Journal on Recent and Innovation Trends in Computing and Communication, vol.3, no. 2, pp.680-683.

[16] Uma Pavan Kumar Kethavarapu and B. Lakshma Reddy. 2014. Data Warehousing Security Encapsulation with Bitmap Indexing Mechanisms. International Journal of Emerging Technology in Computer Science & Electronics, vol.1, no. 11, pp.10-13.

[17] Vuppu Shankar and CV Guru Rao. 2014. Cache based evaluation of iceberg queries. In International Conference on Computer and Communications Technologies (ICCCT), IEEE, pp. 1-5.

[18] M. Laxmaiah, K.Sunil Kumar, A. Govardhan, and C.Sunil Kumar. 2013. A Priority Queue Approach to Evaluate Aggregate Queries Efficiently. WAIMS (World Academy of Informatics and Management Sciences), vol. 2, no. 3, pp. 2278-1315.

[19] He B, Hsiao HI, Liu Z, L. Huang Y, and Chen Y. 2012. Efficient Iceberg Query Evaluation Using Compressed Bitmap Index. Knowledge and Data Engineering, IEEE Transactions on, vol. 24, no.9, pp. 1570-1583.

[20] V Chandra Shekhar Rao, and P. Sammulal. 2014. Efficient iceberg query evaluation using set representation. India Conference (Annual IEEE (2014), pp.1-5.

[21] Marcus Jurgerns, Hans-J. Lenz. 2001. Tree based indexes versus bitmap indexes: A performance study. International Journal of Coopertive Information Systems, vol. 10, no. 3, pp.355-376.

[22] Kesheng Wu, Ekow Otoo and Arie Shoshani. 2004. On the Performance of Bitmap Indices For High Cardinality Attributes. VLDB, pp. 24–35.

[23] Vuppu Shankar and CV Guru Rao. 2013. A Density based Priority Queue Strategy to Evaluate Iceberg Queries Efficiently using Compressed Bitmap Indices. International Journal of Computer Applications, vol. 67, no. 21, pp. 39-44.

[24] Uma Pavan Kumar Kethavarapu, Dr.Lakshma Rddy Bhavanam, and Sreedevi.S. Erady. 2015. Improvement of query processing speed in Data warehousing with the usage of components-Bitmap Indexing, Iceberg and Uncertain data. International Conference on Current Trends in Advanced Computing (ICCTAC-), pp. 1-5.

[25] Ke Wang, Yuelong Jiang, Jeffrey Xu Yu, Guozhu Dong, and Jiawei Han.2003. Pushing aggregate constraints by divide-and-approximate. In Data Engineering, Proceedings. 19 th, IEEE, International Conference, pp. 291-302.

[26] Ke Wang, Yuelong Jiang, Jeffrey Xu Yu, Guozhu Dong, and Jiawei Han. 2005. Divide-and-approximate: A novel constraint push strategy for iceberg cube mining. IEEE Transactions on Knowledge and Data Engineering, vol.17, no.3, pp.354-368.

[27] M. Laxmaiah, K. Sunil Kumar, Dr. A. Govardhan, and Dr. C. Sunil Kumar. 2013. An Approach to Evaluate Aggregate Queries efficiently using Priority Queue. International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), vol. 2, no. 3, pp.341-344.

[28] Every Day Big Data Statistics [online] Available: www.vcloudnews.com/every-day-big-data-statistics.

[29] Ying Mei, Kaifan Ji, and Feng Wang. 2013. A Survey on Bitmap Index Technologies for Large-Scale Data Retrieval. In 2013 6th International Conference on Intelligent Networks and Intelligent Systems (ICINIS), pp. 316-319.

[30] Elizabeth O'Neil, Patrick O'Neil, and Kesheng Wu. 2007. Bitmap index design choices and their performance implications. In Database Engineering and Applications Symposium, 11th International, pp. 72-84.

[31] Raymond T. Ng, Alan Wagner, and Yu Yin. 2001. Iceberg-cube computation with PC clusters. In ACM SIGMOD Record, vol.30, no.2, pp. 25-36.

[32] Yossi Matias and Eran Segaly. 1998. Partitioning based algorithms for approximate and exact Iceberg Queries, pp. 1-30.

[33] Pgukkuo B. Gibbons and Yossi Matias. 1998. New sampling-based summary statistics for improving approximate query answers. In ACM (1998), vol. 27, no. 2, pp. 331-342.

[34] Pei Lee, Lajs V.S. Lakshmanan, and Evangelos Milios. 2014. CAST: A Context-Aware Story-Teller for Streaming Social Content. In Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, pp. 789-798.

[35] Kyu-Yyu Whang, Brad T. Vander-Zanden, and Howard M. Taylor. 1990. A linear-time probabilistic counting algorithm for database applications. ACM Transactions on Database Systems (TODS), vol. 15, no.2, pp. 208-229.