Asian Journal of Information Technology 15 (15): 2671-2678, 2016
ISSN: 1682-3915

Efficient Query Processing of Streaming Data Using Dynamic Heuristic Optimization Approach

[1]M. Ananthi and [2]M.R. Sumalatha
[1]Department of Information Technology, Sri Sairam Engineering College, Chennai, India
[2]Department of Information Technology, Anna University, Chennai, India

Abstract: Data stream management system is used to retrieve and process real-time perpetual data. Information available over web constitute of both static and dynamic data. Semantic illustration will be optimum to process such online streaming information efficiently. Continuous monitoring will be required in the case of real world data which are transient and dependent on both time and location. Effective processor is needed to process large amounts of time-dependent data. Efficient query processing can reduce the execution time and improve accuracy of streaming data. In this research, an efficient query processor is proposed using a Dynamic Heuristic Optimization (DHO) technique. The efficient retrieval of data can be achieved by applying dynamic optimized query plan. Various static and dynamic queries are executed and evaluated. The experimental analysis carried out shows DHO outperforms when compared to existing system.

Key words: Database systems, data processing, query processing, streaming data, real time systems

INTRODUCTION

Data Stream Management Systems (DSMS) is employed to retrieve and process streams of real-time data which varies from database systems in terms of data and query processing. Data are persistent and queries are transient in DBS whereas in DSMS data are transient and queries are continuous. Data streaming consists of infinite number of tuples which are ordered by their timestamp. It can be applied in various areas such as sensor based applications, error log mining, network traffic analysis, telephone call records in web servers, online financial applications, social networking applications and other applications where the streaming data is based on timestamp. It is very difficult to process time-variant infinite number of data because processing queries of real time systems needs additional self joins than database management system. The cardinality estimation for join queries is also challenging and complicated. As an overall perspective from the prevailing systems, efficient query processing is required for real-time, dynamic data. In this research, it is proposed to design an efficient query processor to process real time streaming data. This can be achieved by optimizing and efficient query processing over the streaming data using dynamic programming and heuristic search technique. The proposed research aims to achieve the following:

- Reduction of multiple self-joins by optimizing the query that can help to achieve the efficiency
- Reduction of intermediate results storage by pipelining the query processing further optimize cost
- Evaluation of various simple, complex and streaming queries aimed to analyze the execution time and accuracy of proposed system

Literature review: Query processing and query optimization is one of the challenging issues in data stream management system (Harth *et al*., 2014). Processing continuous query over incoming data streams is a challenging task (Bab Cock *et al*., 2002). So, it is essential to develop an adaptive query processor to improve the system performance. Existing system provide the challenging issues in processing streaming data (Golab and Ozsu, 2003). Dynamic programming algorithm using greedy approach for uncertain data streams has been implemented. Intermediate results are stored and used for further processing in this approach. Storing intermediate results is a materialized approach which may lack in space and performance. However, pipeline technique is appropriate for streaming data to enhance the performance (Chen *et al*., 2013). A scalable architecture was provided to process complex event processing on Linked Stream Data by partitioning the query into operator and data stream. This was implemented as parallel process using pipelining technique called PIPEFLOW. But, not executed with dynamic and uncertain data (Saleh *et al*., 2015).

An extremely scalable and elastic Stream Processing Engine has been implemented to detect fraud calls within the telephone description records in real time. The

Corresponding Author: M. Ananthi, Department of Information Technology, Sri Sairam Engineering College, Chennai, India

Stream Processing engine has been implemented in shared-nothing clusters using parallelization approach to attenuate the distribution overhead. Dynamic programming and greedy approach applied for query processing. No optimization techniques followed, still lack in scalability (Gulisano *et al*., 2012). A formal framework has been proposed to represent windows in continuous queries over data streams and properties of query optimization have been discussed. Algebraic representation of windows, several window types and properties of windowed operators has been mentioned. Dynamic optimization technique has been developed by eliminating intermediate results. A semantic real-time stream management system is proposed. System performance is not monitored in this approach (Rodryguez *et al*., 2009). The RDF compatible model have been developed with semantic stream to store and process triples with efficiency. Query optimizer has not been developed and the query hasn't been evaluated. Data collected from numerous heterogeneous sensor data. A standard model has been developed to cover the heterogeneity by R2RML. Query rewriting and data translation has been done by $SPARQL_{stream}$ approach. Metrics of four query engines SNEE, GSN, Pachube and Esper has been mentioned. Simple queries have been considered whereas complex queries are not considered for optimization (Calbimonte *et al*., 2012). Numerous self-joins of physical relational data structure for RDF triples are involved. Statistics Maintenance and search space makes the optimization of query plan at the run time terribly pricey (Danh *et al*., 2012).

Semantic based optimization is required for continuous updating and monitoring stream of data (Spanos *et al*., 2012). Efficient indexing and discovering mechanism are required for efficient storage, data handling for large volume of data that publishes and access semantic data in large distributed and dynamic environments. A new indexing approach CKDB-tree was proposed and processed with General Purpose Graphics Processing Unit (GP-GPU) to reduce storage cost and to handle dynamic continuous queries over streaming data. Queries were split into cells and KDB-tree was constructed. Repeated partitioning of particular cell will lead to space overhead, thus reduces the efficient retrieval (Deng *et al*., 2015). With respect to the fundamental concept of database systems (Elmasri and Navathe, 2007) it is better to avoid Cartesian product while joining more than one data which takes close to 1 million seconds to process 1 GB data. Jeff Edmonds Dynamic Programming algorithm and Heuristic Search Method (Skiena, 2008) was given which helps to develop the proposed approach (Edmonds, 2000). To overcome the difficulties in existing system, it is proposed to design an efficient query processor for streaming data with DHO (Dynamic Heuristic Optimization).

MATERIALS AND METHODS

Stream query processor: Streaming data in real time systems are represented in large volume of tuples. The ongoing work of data stream management system requires efficient query processing to process the dynamic queries. Efficient query processing is needed to access this huge number of tuples. The work proposes to develop an efficient query processor to retrieve and process streaming data. This can be achieved by optimizing the query, reducing the number of joins and applying proficient reasoning over the query. A stream query processor is proposed to achieve fast retrieval of streaming data. Stream query processor architecture is represented in Fig. 1. Streaming data e.g., online stock market readings is retrieved and it is dynamically given as input to processor engine. The incoming streaming data is collected in the form of tuples via web interface. Streams of data are controlled by a sliding window that is stored in the buffered queue. Investors request query about the stock details via web interface which is handled by query processor. Query processor operations can be split into two stages; input query is parsed by query parser and optimized by optimizer. Optimized query is executed by searching in the index using the pipelined approach and by applying reasoner over it. The reasoner chooses dynamic optimal plan. Static data are also used to execute the query. Static data is collected from the metadata which is stored in the database. Optimized query plan is generated dynamically using heuristic dynamic program technique. The results of relevant information are displayed in terms of tuples.

Generally, queries are processed by first scanning it by identifying the token string and then parse the query into syntax tree. Parsed tree has been validated by semantic check and it checks for any constraints specified to process the query. After validating the query, the logical query plan is constructed which is also called as intermediate query representation. This logical representation is to represent the query in terms of relational algebra. A specific set of rules are used to reduce the overhead in data processing. Finally, the relational algebra with set of rules is given to query

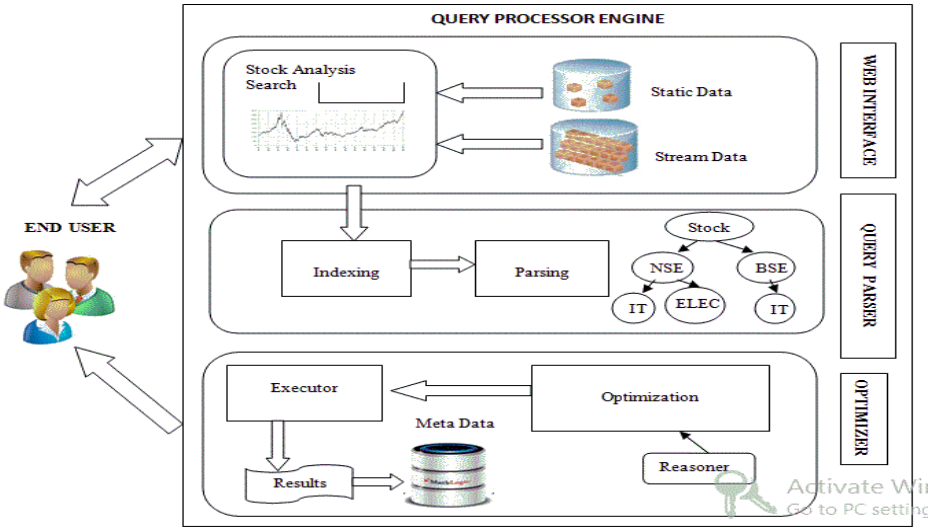

Fig. 1: Stream process architecture

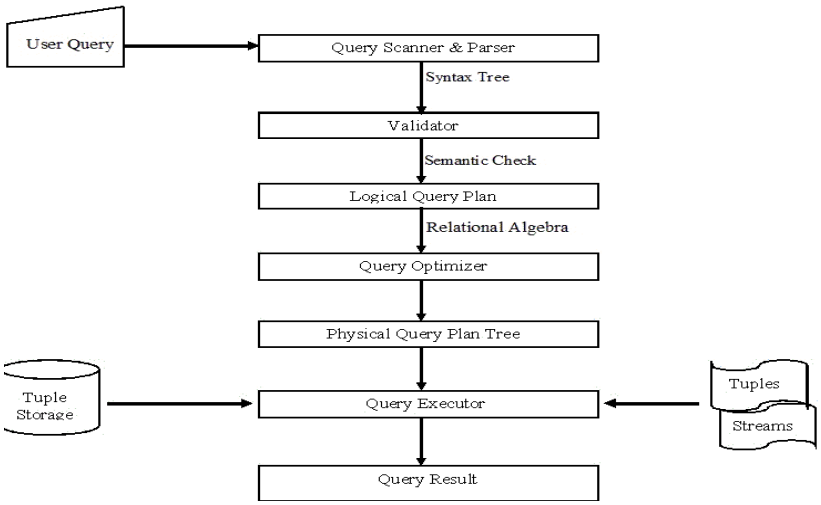

Fig. 2: Query processor

optimizer. The query optimizer converts the logical query plan to physical query plan tree which is the query execution plan. Queries are executed based on the execution plan as shown in Fig. 2. Input to process the query has been fetched as tuples either statically or dynamically based on the query requirement. Finally, it produces the query results in terms of tuples.

Dynamic query plan: Dynamic programming and heuristic search can be used to prune the data for increasing the efficiency of query processing. Materialized way is non-pipelined process executed by pre-computed query plans which consumes high cost estimation compared to pipelined execution. The intermediate results are stored and used for further optimization. Streaming data does not require storing intermediate results that consumes more space and cost effective. Streaming data can be re-optimized by changing the query plan at any stage. Pipelined execution improves the performance for optimized plan as proposed in Dynamic Heuristic Optimization (DHO) approach which is given in algorithm 1.

Algorithm 1: Dynamic Heuristic Optimization (DHO) algorithm:

Input: Query (Q, T); Data (D, T), Sliding Window W
Result: Optimized Query result R
1 Let the minimum cost $Cost_{min}$ to be 0
2 Set the execution time as-1

3 for each timestamp T in [W_1,W_2…W_n] do
4 Retrieve the tuples based on window size W//FIFO queue
5 Q^T←W(S,ts,T)
6 If Q^p.size() == 0 then
7 Return 0
8 Else
9 Remove duplicates
10 Optimize_plan()
11 End if
12 End for
13 Return R

Input is the query Q which is given by the user continuously. In the algorithm Q represents the query; T indicates the timestamp which is required for streaming the query. Data is given as D. W is the sliding window by which the incoming streaming tuples are controlled by splitting the tuples in each window where W_iW. Each W_i consists of fixed number of tuples. Various sliding window models can be used. They are tuple-based sliding window, time-based window and range-based partitioned window. Depending on the requested query the type of window is chosen. If the query is specified without timestamp then it is considered as a static query (line 3). The algorithm calls the sub-routine optimize-plan to find the optimized plan for the given query Q (line 10). If it is a timestamp-based query then it retrieves the number of tuples limited by the window and it is stored temporarily in Q^T (line 5) and further is sent to query optimization(line 10). Then, after removing the duplicates it calls sub-routine Optimize-plan. Duplicates are removed by selecting only distinct values by applying the inference rule. W(S, ts, T) is the sliding window which has the Stream S with the intermediate timestamps ts where. If no tuples are received within the time period specified, then optimization is not required that results in no cost and processing time [line 7].

The subroutine optimize-plan is signified in algorithm 2. It is based on dynamic programming optimization technique with heuristic search. Entire query is split into sub queries for easy processing. Optimal plan is applied to each sub query and optimal edge is identified as E (V_i, V_k) (step 7) that finds the next optimal solution by recursively applying the same. Cost of each optimal plan is computed and stored in W(V_i, V_k) added with each suboptimal solution (step 8). Finally minimal cost and solution is achieved and returned (step 11-16). Query processing cost is challenging to process dynamic streaming data. However, cost can be calculated based on the meta data and incoming data in the buffer. According to DHO, no intermediate results are stored. Cost is calculated based on number of relations referred, number of matching tuples and number of attributes required and size of each attribute. In addition to that processing time of number of joins and aggregate functions are considered.

Algorithm 2: Optimize-plan:

1 Input: Query Q
2 Partition the query into sub query
3 Q^-←{SQ_1, SQ_2, …, SQ_n}
4 C←0
5 Sol←{?}
6 While Q^- is not empty do
7 For every SQ_i do
8 Sol_i←E(V_i,V_k)+opt_{subsol}//V_k-intermediate vertex
9 C_i←W(V_i, V_k)+opt_{cost}
10 End For
11 //optimal solution and optimal cost
12 Sol←ÓSol$_{i..n}$
13 C←ÓC$_{i..n}$
14 end While
15 Optsol←{k_{min}(Sol)}
16 Optcost←Min(C)
17 Return (Optsol, Optcost)

Let each relation be denoted by R_i. Let $1 \le i \le N$, N is the number of relations. Number of tuples in each relation R_i is represented as $|R_i|$. Let b_{jk} be the size of each jth attribute of kth relation. Total size of selected attributes of kth relation is:

$$Tsk = \sum (bpk + bqk + ...).mk \quad (1)$$

Where:
bpk, bqk = The size of each selected attributes varying as bp, bq
mk = The number of relations varying with m

Selection cost Sc with selected attributes in kth relation based on some condition α is denoted as:

$$Sc = Tsk \times |Rak| \quad (2)$$

Where:
Tsk = The total is size of selected attributes of kth relation
Rak = The number of selected tuples of kth relation with condition α

Join cost Jc of two relations is specified as:

$$Jc = Sck \times Scm \quad (3)$$

where, Sck and Scm is the selected cost of kth relation and mth relation. Let Ac be the cost of aggregate function. Overall cost of the query is calculated from the above Eq. 2 and 3:

$$Oc = \sum_i (Sci + Jci + Aci) * \prod js \quad (4)$$

Where the selected attributes required to project in the final result is Πjs with the sum of various number of relations in the final result which varies over 'i'. Optimal cost is calculated from Eq. 4 as:

$$Opt_{cost} = \min(Ock) \quad (5)$$

where, Ock is the overall cost of each plan in which k varies dynamically.

RESULTS AND DISCUSSION

Implementation analysis: Real time systems have been taken for experimentation. Stock market is one of the applications that use streaming data. Live value of the stock gets varied in seconds based on the change in company's value which requires the company details which are static linked with the live market. Both static and dynamic (stream) queries are considered for execution. All experiments are conceded in a machine with an Intel® Core i5 M480 2.67 Ghz CPU and 4GB RAM×64-based processor. API, is developed for query processing which retrieves live data from the web and processes the requested query. Live readings are collected from the online stock web site http\\:www.money.rediff.com. An application named Tray App. is developed which records the live stock market values and continuously updates the current stock value of each company. This updating is done every time when there is a small change in the live values. Queries are framed to analyze the system performance. Information which cannot be retrieved directly through stock website is considered for execution. Following sample queries are executed and evaluated:

- Query 1: Retrieve the stock values observed once in 30 sec
- Query 2: Detect the list of companies that are in high contingency position before closing time of stock market
- Query 3: Display the minimum and maximum share value of any company on the given day
- Query 4: Display the top 10 list of companies at a particular date and time
- Query 5: Display the list of companies that has undergone a tremendous growth in recent times
- Query 6: Predict the profit on future investments between different sectors or different companies
- Query 7: Detect the list of gainers and losers at generic time

Values are steered from TrayApp application designed to retrieve live values from web for Query 1. Company's current stock value is updated whenever there is a change in the live stock value. Data updation is done with dynamism since the change timing of stock values is unpredictable. So, TrayApp monitors the live readings which are a streaming data and performs the updation dynamically. Query 2 is designed to analyze the companies that are in a critical position at the closing time of market which is not available in real time. Companies sell their share values at an available rate (preferably not for profit) nearby the closing session of the stock market which helps to provide awareness to the investors whereas investing in those lists of companies which are at risk. Readings are monitored at the end of the day. Query 3 is used to display the minimum and maximum share value of any company at a generic date and time which is highly used by the shareholders to know about the position of the company at any given time. Query 3 is a complex query designed to test the performance of dynamic optimal plan.

Query 4 is designed to identify the companies that acquiretop 10 positions at a generic date. User can know about the ranks of the companies which will be used to make decisions on investing in the best companies. Top positions are not earned by a particular date in existing system. Companies are divided into groups namely group A, B, Z and T based on the nature of stocks and their growth. For example, Group A contains top 30 companies contributing to the stock market. Group B contains the list of all companies in stock market. Group Z contains the companies that have undergone tremendous growth in recent times. This is achieved by Query 5. The end user can know about the list of companies that have achieved tremendous growth by using this query. General users are not aware of the group details. So, this query is used for new investors and also in testing the performance. Query 6 is used to predict the profit for future investments. Only two specified companies from the same sector can be compared in the existing system. However in this query, comparison between various companies that may be from same sector or different sectors can be achieved. Stock values of the past one year are given as metadata and the predictive analysis is done based on those data. Query is chosen to test the performance of a complex query. Query 7 retrieves the top gainers and losers in the stock market which is already retrieved through web; it is used here to evaluate the system performance.

Optimal query plan: The queries are executed as specified in algorithm 1 and 2. Query Plan is chosen dynamically. The plan which has minimal execution cost is considered as optimal plan. The best plan to optimize query processing is to first perform the select operation with join condition and then project the result. This can be represented as a bushy tree which reduces intermediate results and allows asynchronous execution of sub trees. For example the optimal plan for complex query 3 will be defined in different ways.

Execution cost is evaluated for each plan. Plan 3 has an optimal cost. So, Plan 3 is chosen for execution. It is shown in Fig. 3.

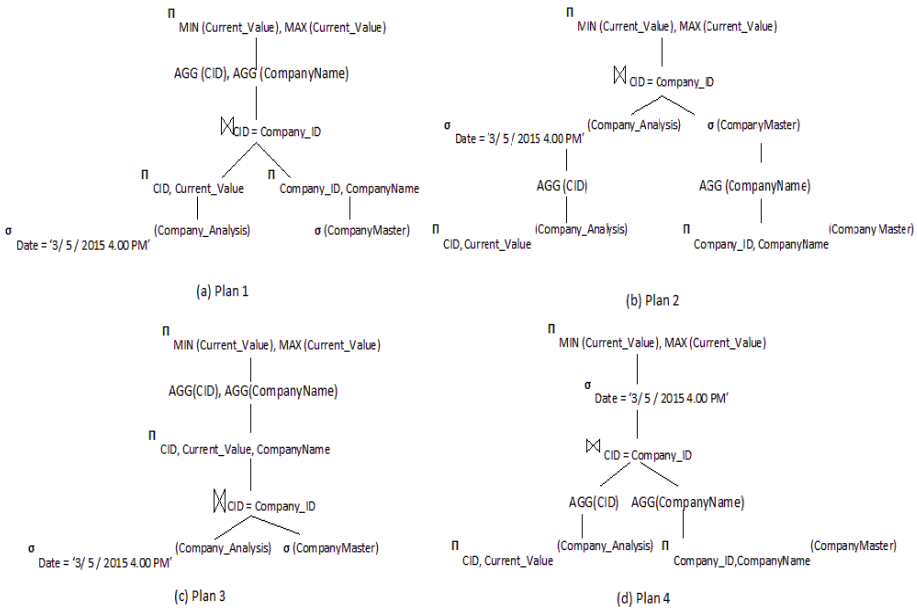

Fig. 3: Various execution plans for query

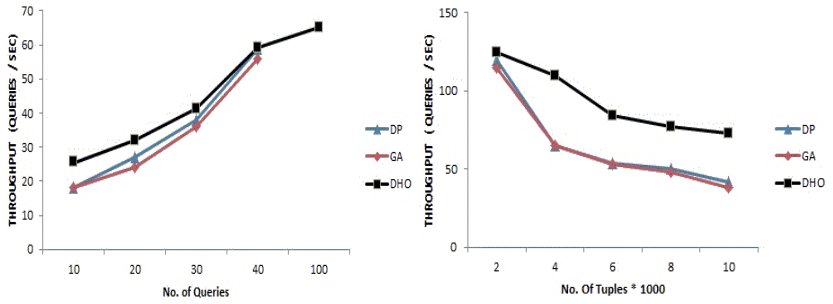

Fig. 4: Throughput performance with various number of queries and number of tuples; DP-Dynamic Programming Approach; GA-Greedy Approach; a) Throughput vs. no of queries; b) Throughput vs. no of tuples

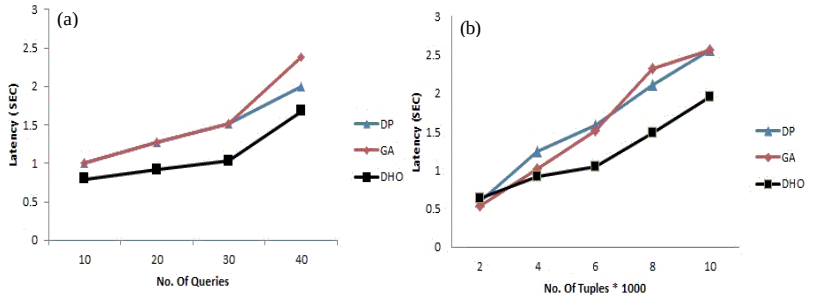

Fig. 5: Latency performance with various number of queries and number of tuples; DP-Dynamic Programming Approach; GA-Greedy Approach; DHO-Proposed Dynamic Heuristic Optimization approach; a) Latency vs. no of queries; b) Latency vs. no of tuples×1000

Relational algebra of plan 3 for query 3 is written as:

- $T_1 \leftarrow \sigma_{\text{Date = '3/5/2015 4.00 PM'}}$ (Company_Analysis)
- $T_2 \leftarrow \sigma$ (Company master)
- $T_3 \leftarrow T_{1 \text{ Join CID = Company ID}}\ T_2$
- $T_4 \leftarrow \pi_{\text{CID, Cname, Current\_Value}}\ (T_3)$
- $T_5 \leftarrow \pi_{\text{CID, Cname}}\ (T_4)$

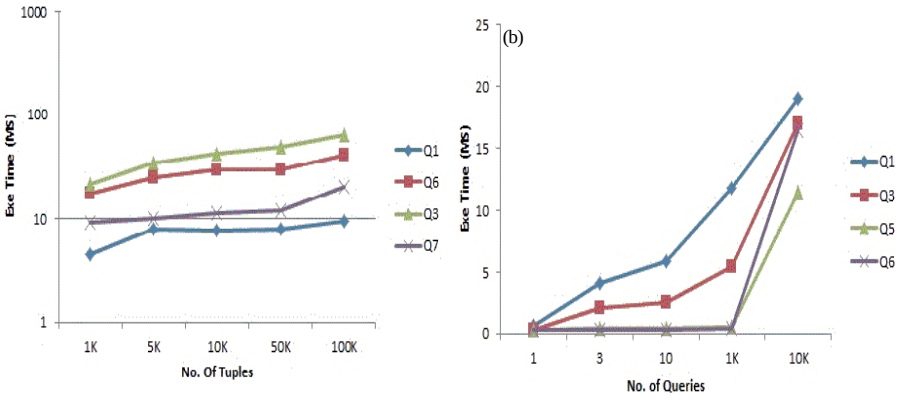

Fig. 6: Query execution time with various number of queries and number of tuples; DP-Dynamic Programming Approach; GA-Greedy Approach; DHO-Proposed Dynamic Heuristic Optimization approach; a) Execution time vs. no of queries; b) Execution time vs. no of tuples

- $T_6 \leftarrow \pi_{\text{MIN (Current\_Value), MAX (Current\_Value)}} (T_5)$

Query 3 is executed in SQL as follows:

```
Select Company_Analysis. Company ID, Company Master. Company Name, MIN (Current_Value) as minimum, MAX (Current_Value) as maximum
from Company_Analysis INNER JOIN Company Master
ON
Company_Analysis. Company ID = Company Master. Company ID
where A_Date = "'+date+'"
group by Company_Analysis. Company ID, Company Master. Company Name
```

Above mentioned query is to find the minimum and maximum stock value and their company details on particular date and time. Cost for each plan is evaluated and measured in terms of number of tuples retrieved at each step and their execution time is measured. Various execution plans are represented in Fig. 3a-d among which the plan 3 provides the optimal cost. They may have any number of plans to execute the same query. Four plans are suggested here for evaluation. All other queries are executed in a similar way.

Evaluation and results: Real data sets which are retrieved from web are considered for execution. Stock data is a dynamic data that depends on time. There is infinite number of stock values which varies from time to time. System performance is measured with various parameters such as throughput, latency and execution time. Performance of proposed algorithm is compared with existing algorithms DP (Dynamic Programming) and GA (Greedy Approach) that are discussed in existing system. Throughput and Latency performance is signified in Fig. 4 and 5. Throughputs achieved at the systems GA and DP are almost the same is shown in Fig. 4a. DHO proposed approach improves the performance by nearly 5-7%. Throughput is high even when the number of queries is >100. Throughput of DHO also increases as the number of queries increases. Explained method is better than existing methodologies DP and GA. High throughput is achieved by increasing the number of tuples as shown in Fig. 4b. Even the throughput is reduced while increasing the number of tuples but it is better than DP and GA. Latency of DP and GA are almost the same but small variations occur when number of queries and tuples are increased. Latency of DHO is slower than DP and GA as shown in Fig. 5a. Latency slightly increases when the numbers of queries are raised which is only 2% increase which is lesser than DP and GA. Latency is measured by executing number of queries per second by increasing number of tuples. Less latency is achieved than DP and GA as shown in Fig. 5b. Various numbers of queries are executed and their performance is monitored. Static, dynamic and complexqueries are considered for execution. Query 1-7 is represented as Q1-Q7 respectively. The performance graph for Q1,Q3,Q6 and Q7 by varying number of queries and number of tuples are shown in Fig. 6a and b. Q1 is a dynamic query which retrieves current stock value continuously and produces stream of data which varies in time. Execution time of Q1 is higher than other queries, in which stock value and its time should be monitored dynamically as shown in Fig. 6a. Q3 computes the minimum and maximum value every day. It is a complex query and it has taken time to perform some computation. So, the execution time of Q3 is higher than Q6 and 7. Q6 and 7 almost have the same execution time. Both Q6 and 7 are complex queries and refer to the metadata which is static. Number of comparisons and computations are also increased when the n number of queries is increased which leads to a higher execution time. The execution time of each query by varying number of tuples is shown in Fig. 6b. Same set of queries Q1, Q3, Q6, Q7 are measured in terms of number of tuples retrieved per millisecond. Execution time for Q3 and Q6 is higher than Q1 and Q7. Both are complex query and must perform more number of

Table 1: Execution time of different queries

Queries/exe. time	1 K	5 K	10 K	50 K	100 K
Q1	4.6	7.9	7.8	7.93	9.4
Q3	21.4	35.1	42.67	48.93	64.76
Q6	17.4	25.4	29.7	30.2	41.2
Q7	9.3	10.12	11.35	12.1	20.3

comparisons and computations. Q7 retrieves more number of records than Q1 so execution time of Q7 is higher than Q1. The system performance is improved by applying dynamic heuristic approach. Execution time by varying number of queries that are measured in milliseconds is shown in Table 1.

CONCLUSION

An efficient query processor with DHO approach is designed by optimizing and proper reasoning of streaming data. Optimization is used for the fast execution and retrieval of query processing. Reasoner is added to reduce number of joins in the intermediate results. This is achieved by dynamic programming and heuristic search technique. Real-time stock market readings are taken as streaming data for experimentation. Various types of queries (simple, complex and multi queries) are executed and assessed. The performance of the proposed work is achieved by improving the execution time and throughput. Throughput is increased by 13% but Latency is reduced by 4% when compared to the existing system. In future, it is proposed to improve the performance by providing alternate indexing and optimizing of query processor.

REFERENCES

Babcock, B., S. Babu, M. Datar, R. Motwani and J. Widom, 2002. Models and issues in data stream systems. Proceedings of the 21st Symposium On Principles Of Database Systems ACM SIGMOD-SIGACT-SIGART, June 2-6, 2002, ACM, New York, USA., ISBN:1-58113-507-6, pp: 1-16.

Calbimonte, J.P., H.Y. Jeung, O. Corcho and K. Aberer, 2012. Enabling query technologies for the semantic sensor web. Int. J. Semantic Web Inf. Syst., 8: 43-63.

Chen, T., L. Chen, M.T. Ozsu and N. Xiao, 2013. Optimizing multi-top-k queries over uncertain data streams. IEEE. Transac. Knowl. Data Eng., 25: 1814-1829.

Danh, L.P., X.P. Josiane and M. Hauswirth, 2012. Linked stream data processing reasoning web: Semantic technologies for advanced query answering. Lect. Notes Comput. Sci., 7487: 245-289.

Deng, Z., X. Wu, L. Wang, X. Chen and R. Ranjan *et al*., 2015. Parallel processing of dynamic continuous queries over streaming data flows. IEEE. Transac. Parallel Distrib. Syst., 26: 834-846.

Edmonds, J., 2008. How to Think about Algorithms. Cambridge University Press, Cambridge, UK., Pages: 318.

Elmasri, R. and S.B. Navathe, 2007. Fundamentals of Database Systems. 5th Edn., Addison-Wesley, USA., ISBN-13: 9780321369574, Pages: 1139.

Golab, L. and M.T. Ozsu, 2003. Issues in data stream management. ACM. Sigmod Record, 32: 5-14.

Gulisano, V., R.J. Peris, M.P. Martinez, C. Soriente and P. Valduriez, 2012. Streamcloud: An elastic and scalable data streaming system. IEEE. Transac. Parallel Distrib. Syst., 23: 2351-2365.

Harth, A., H. Katza and S. Ralf, 2014. Linked Data Management-Emerging Directions in Database Systems and Applications. Chapman and Hall/CRC, London, UK., Pages: 485.

Rodryguez, A., R. McGrath, L. Yong and J. Myers, 2009. Semantic management of streaming data. Proc. Semantic Sens. Networks, 2009: 80-95.

Saleh, O., S. Hagedorn and K.U. Sattler, 2015. Complex event processing on linked stream data. Datenbank Spektrum, 15: 119-129.

Skiena, S.S., 2008. The Algorithm Design Manual. 2nd Edn., Springer, Berlin, Germany, ISBN: 978-1-84800-069-8, Pages: 709.

Spanos, D.E., P. Stavrou, N. Mitrou and N. Konstantinou, 2012. SensorStream: A semantic real-time stream management system. Int. J. Ad Hoc Ubiquitous Comput., 11: 178-193.