International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Multiple Query Processing with Group Filters on Data Streams

Nam Hun Park[1], Kil Hong Joo[2]

Submitted: 06/06/2022 **Accepted:** 10/09/2022

Abstract-In the environment of stream data, real-time data is continuously and continuously generated, but a plurality of continuous queries are statically registered and performed on real-time data. Therefore, it is important to find only the queries that can be processed on the arrived data, and to quickly execute only the queries to reduce the burden of query execution on the system. In order to solve the problems of the existing multi-query processing method, in this study, the order of attributes is determined according to the selection rate of attribute conditions. Real-time performance can be optimized by finding the optimal attribute order in the group query. Through experiments, it has been demonstrated that excellent performance is shown even when the number of properties is large.

Keywords*: Data Stream, Group Filters, Multiple Queries*

1. Introduction

Many existing data systems receive real-time data, biometric information, and behavioral information from sensors based on a real-time environment and monitor or analyze them. In particular, most medical application environments use simple and passive monitoring techniques for data. However, a real-time monitoring method for various conditions such as continuous trend change as well as fragmentary analysis is required.

The data stream is real-time, continuous, fast, and infinitely new data is generated. Because of this feature, it is not a one-time query for all data, but a continuous query format in which a query is registered in advance and the execution result is notified whenever data is generated or periodically[1,2]. One-time queries are executed once and removed from the system, but continuous queries are continuously executed each time a data stream arrives and results are accumulated, and the query is repeatedly processed every time a new window starts. Since continuous queries are stored in a static state in the data stream, performance can be improved by creating a group query plan that shares the operation results or by using group processing methods rather than executing individual queries for many continuous queries. In addition, quick judgment on the infinitely fast influx of data also has a big impact on performance. In this paper, we propose a method for fast real-time multi-query processing by analyzing multiple query filters for more efficient query search in a data stream real-time monitoring system. In the data stream environment, it is effective to group many selection conditions through indexing into various data structures. Most of the group processing is grouped by attribute, and group filters are created as many as the attributes used in the query, and there are many performance differences depending on the execution order. Therefore, to determine the execution order, we propose a method for determining the low-cost order by considering the selectivity of the condition and the number of queries for which attributes are used.

The composition of the thesis is as follows. Section 2 describes the related works on query processing and optimization. Section 3 describes the method of group filters on the query conditions and how to order the filters. Section 4 analyzes the performance through comparison and finally, Section 5 describes the conclusion and future research tasks.

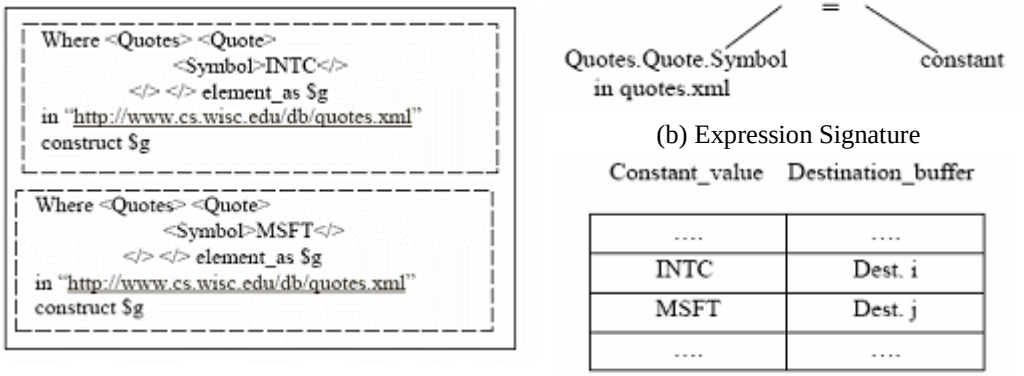
2. Related Works

NiagaraCQ[3] is a system for processing query results for

[1] *Anyang University, Korea*
ORCID ID: 0000-0002-3716-5760
[2] *Gyeongin National University of Education, Korea*
ORCID ID: 0000-0002-5326-8495
**Corresponding Author E-mail: khjoo@ginue.ac.kr*

XML streams and monitoring changes in web documents of interest. It was developed for continuous query processing as part of the Niagara project for XML stream processing. By analyzing the query format for multiple queries, queries with similar expressions are grouped to share calculations. Query conditions with the same expression form with the same properties and operators are created in a structure called a group constant table. Query processing is processed in the form of a join between the group constant table and data, in which multiple selection conditions are stored.

In NiagaraCQ, as Figure 1, a condition with the same grammar or expression structure in a query expressed using a query language called XML-QL is created as a group constant and data is joined with this table. The processing result is shared through the split operation by distributing the query result. When processing the selection condition and group processing of the join condition, the optimization method according to the order is used, and when a query is added, the query condition is dynamically regrouped.



(a) XML-QL

(b) Expression Signature

(c) Group Constant Table

Figure 1- Group Query Processing in NiagaraCQ

CACQ[4] is a system that dynamically changes the order of operators continuously during execution without executing a fixed query plan. CACQ indexes the selection conditions in a way called a grouped filter, which classifies the selection conditions of a query as attributes and groups them into operations again. The group filter is created as many as the number of attributes appearing in the selection query and has two hash structures and two AVL tree data structures internally[5,6]. The comparison value of the selection condition used in the query is separated and indexed according to the type of operator. When a new query is registered, if there is a group filter of the attribute used in the selection condition, it is added to the corresponding group filter. If not, a new group filter is created. The data tuple is processed through the group filter, and the result is shared by all queries using that attribute.

3. Adaptive Query Processing

Multiple continuous queries applied to the data stream are classified into group filters based on the query conditions. The group filter is a selection module for real-time tuples. The real-time data tuple is routed to a group filter composed of continuous query conditions[7,8]. When there are multiple selection modules, group filters can be randomly selected or filter counting can be performed.

Random routing performs random routing as in the term, and filter counting increases filter counting by giving a tuple to the group filter, and decreases the counting number again when the group filter removes the tuple. Therefore, a large number of counting means that many tuples have been removed, and an efficient group filter can be selected by selecting a large number of tuples[9].

The existing multiple continuous query filter method repeats unnecessary processing when attributes are not used in all queries. For example, if you have queries like follows:

Continuous Queries : select * from R where R.attr1 = ‘a’ (0.3) and R.attr2=’a’ (0.1)
select * from R where R.attr1 = ‘b’ (0.3) and R.attr2=’b’ (0.1)
select * from R where R.attr1 = ‘c’ (0.3)

If two group filters of R.attr1 and R.attr2 are made and routing is performed to the group filter of R.attr1 first, the R.attr1 filter will accumulate 10% counting of tuples from a probabilistic view. Assuming that 60% of the remaining tuples are routed to R.attr2, 48% of them are filtered. Therefore, in the example, R.attr2 is routed preferentially. In fact, all tuples entering the R.attr2 filter have to go back to R.attr1 to process the condition for the third query. Therefore, the sequence of R.attr1 and

R.attr2 is repeated. The best order is to process R.attr1 first and then proceed to R.attr2 because all data processed in R.attr2 should proceed to R.attr1.

If *n* attributes are used in the selection condition of multiple queries, *n* group filters are created. The order of *n!* is possible for n group filters. From now on, we propose a method for determining how to determine the order.

When there is one query with multiple selection conditions, or when group processing of selection conditions from multiple queries, processing costs are different in different orders. When processing multiple conditions in a single query, if the conditional selection rate considering the interrelationship can be known, the optimal order can be determined by applying the Greedy algorithm. However, if the conditional selection rate is unknown, the order is determined in the order of the selection rate that passes the tuple with the minimum number of unconditionally. Let's find the cost of processing a single query with multiple selection conditions in the selection condition processing sequence. When a single query using *n* conditions processes *t* tuples, if each condition has a processing cost *Ci* and a condition selectivity *Si*, the processing cost *Ct* can be obtained as follows.

$$Ct = t\sum_{i=1}^{n} Ci\prod_{j=0}^{i-1} Sj$$

Assuming that the processing cost *Ci* of each i^{th} condition is the same, it is as follows.

$$Ct = tC\sum_{i=1}^{n}\prod_{j=0}^{i-1} Sj$$

However, the group processing method that integrates the selection conditions of multiple queries is much more complicated than this. The possible order of n group filters in a group filter in which the selection condition is indexed according to an attribute is n! Among them, if the probability of a tuple passing through the group filter in the order *Op* of one order group filter is *Li* and the processing cost *Ci* of i^{th} attribute, the total processing cost for *t* tuples is as follows, similar to a single query.

$$C_t(O_p) = t\sum_{i=0}^{n} C_i\prod_{j=0}^{i-1} L_j$$

Assuming that the processing cost of the group filter is the same, it is as follows.

$$C_t(O_p) = tC\sum_{i=0}^{n}\prod_{j=0}^{i-1} L_j$$

In single processing, the selection rate of the condition always has a fixed value regardless of the order, but in the case of group processing, the probability of passing the attribute selection varies depending on the position. That is, the number of tuples filtered by the attribute selector changes depending on how many queries were made "false" by the previous attribute selectors. Since the processing cost *Ct* is different for each order, if the processing cost of the attribute selectors is assumed to be the same, the best order is the order of checking the minimum attribute selector. Therefore, it is the best order to find the order in which the sum of the products of *Lj* is minimized.

The order of tuples satisfying all conditions does not matter, but when all queries are "False", various performances are shown according to the order. In a single query with multiple conditions connected by “AND”, if one condition becomes “False”, the query also becomes “False” and the corresponding data tuple can be removed. For example, if the total number of queries is t, even if the data tuple makes t-1 queries “False”, if the other query is “True”, the corresponding data tuple cannot be removed. Therefore, if a data tuple that makes all queries “False”, that is, a data tuple that is removed from the system makes *t-1* queries “False”, if the condition of the other query is not checked, it is not removed and continues until the condition is checked. If the condition that answers the single query is at the end of the sequence, the tuple that becomes “False” will unnecessarily check many group filters, so it is better to select the group filter that uses the condition in as many queries as possible first. Advantageous. The attribute utilization means how many queries an attribute is used.

If the set with *n* attributes used in the query is ***A***, then ***A*** = { *A1, A2 ,… , An*}. The property utilization *U* of one property *Ai* of property set ***A*** is determined like follows:

U(Ai) = The number of queries with conditions on *Ai* / The total number of queries

Considering these characteristics, it creates a requirement to quickly remove unnecessary data tuples by first checking the attributes used in all queries, that is, attributes with attribute utilization of 1. If there is no attribute used in all queries, it finds a set of attributes that fulfill one or more conditions of all queries, that is, a requirement that data tuples can be discarded in combination with other attributes. Since the number of elements is a property that must be checked for tuples to be filtered, the smaller the number of elements, the more advantageous.

4. Experimental Results

In the comparison experiment, three types of data sets were used. Each data set has all 20 attributes, and the domain range of each attribute is 0~99. Each data set has 500,000 data tuples and has distribution characteristics as shown in Table 1.

Table 1. Experimental Data Sets

Data Set	Features
#1	Each data value has a uniform distribution
#2	It has a high distribution of median values in a form similar to a normal distribution graph.
#3	A distribution similar to the shape of a pulse, with the first half having low data values and the next half having high data values.

Figure 2 is a performance test for the minimum attribute cover set. The selection rate and utilization of the attribute condition of each attribute were applied in various ways. The order was determined by performing the properties of the minimum property cover set with the highest priority and arranging the rest in alphabetical order. In the series, the worst case is the worst case as a result of executing all possible sequences, the best case is the best case, and the max utilization means the performance by selecting attributes of higher utilizations values first.

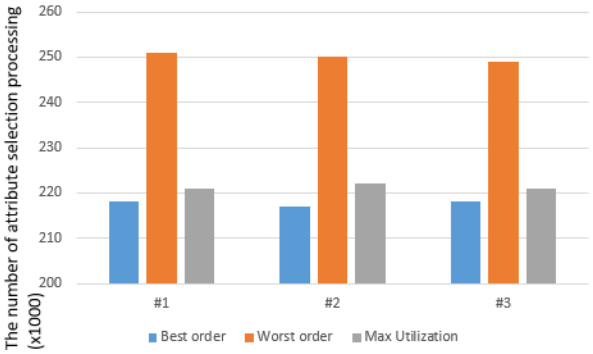


Figure 2- Performance Comparisons by Data Sets

Figure 3 shows the experimental results when the minimum attribute cover set is applied while increasing the number of attributes to 20. Since it takes too much time to perform all sequences from 10 or more attributes, after randomly performing 2000 times, the worst case is set as the maximum processing order and the best case is set as the minimum processing order. The more attributes, the more queries and more conditions, so the difference in performance according to the order becomes clear. The data set was #1. As can be seen from the experiments in Figure 2 and Figure 3, it can be seen that the processing cost can be significantly reduced just by finding the minimum attribute cover set and starting the sequence.

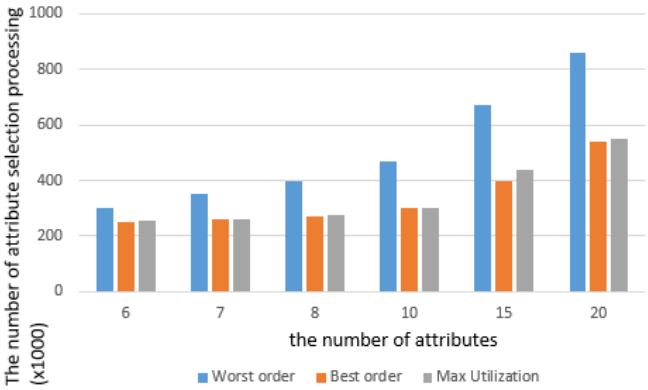


Figure 3- Performance Comparisons with the number of attributes

5. Conclusion

As science and technology advances in the future, the types of sensors constituting the sensor network will increase, the amount of data that the server needs to process instantaneously to provide real-time services will gradually increase, and the queries that users request from the system will also become more complex. Therefore, in this paper, we propose a property selection method that shows excellent performance for a large number of query searches in a real-time environment. The goal of this study is to effectively perform real-time continuous processing for smart medical application services with fast performance in a real-time WBAN environment. Therefore, future research is to develop a complete application prototype by integrating the stream data real-time monitoring technique as an application layer implemented in this paper with the physical layer and upper layer of communication, and to optimize it by performing performance analysis.

References

[1]. Wahab, Raja Azhan Syah Raja, et al., A Method for Processing Top-k Continuous Query on Uncertain Data Stream in Sliding Window Model. *WSEAS Transactions on Systems and Control*, 16 (2021), 261-269.

[2]. Metre, K. V, Location based Continuous Query Processing over Geo-streaming Data. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, 12.1S (2021): 106-114.

[3]. Avhankar, M. S. ., D. J. A. . Pawar, S. . Majalekar, and S. . Kedari. “Mobile Ad Hoc Network Routing Protocols – Using OPNET Simulator”. International Journal on Recent and Innovation Trends in Computing and Communication, vol. 10, no. 1, Jan. 2022, pp. 01-07, doi:10.17762/ijritcc.v10i1.5513.

[4]. Chen, Jianjun, et al., NiagaraCQ: A scalable continuous query system for internet databases. *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, (2000).

[5]. Madden, Samuel, et al., Continuously adaptive continuous queries over streams. , (2002).

[6]. Le-Phuoc, Danh., Adaptive optimisation for continuous multi-way joins over rdf streams. *Companion Proceedings of the The Web Conference*, (2018).

[7]. Katsipoulakis, Nikos R., Alexandros Labrinidis, and Panos K. Chrysanthis, Concept-driven load shedding: Reducing size and error of voluminous and variable data streams. *2018 IEEE International Conference on Big Data (Big Data)*, (2018).

[8]. Kose, O., & Oktay, T. (2022). Hexarotor Yaw Flight Control with SPSA, PID Algorithm and Morphing. International Journal of Intelligent Systems and Applications in Engineering, 10(2), 216–221. Retrieved from https://ijisae.org/index.php/IJISAE/article/view/1879

[9]. Liao, Zhining, Hui Wang, and Gongde Guo, The optimal query plan selection based on the network and remote server analysis. *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*, Vol. 7, (2004).

[10]. Rosemaro, E. . (2022). Understanding the Concept of Entrepreneurship Management and Its Contribution in Organization. International Journal of New Practices in Management and Engineering, 11(01), 24–30. https://doi.org/10.17762/ijnpme.v11i01.159

[11]. Liang, Y., Lee, J., Hong, B., & Kim, W. C., *Real-time Processing of Rule-based Complex Event Queries for Tactical Moving Objects. In COMPLEXIS*, pp. 67-74), (2019).

[12]. K, S., & srinivasulu, T. (2022). Design and Development of Novel Hybrid Precoder for Millimeter-Wave MIMO System. International Journal of Communication Networks and Information Security (IJCNIS), 13(3). https://doi.org/10.17762/ijcnis.v13i3.5096

[13]. Babu, Shivnath, and Jennifer Widom, *Continuous queries over data streams. ACM Sigmod Record, 30.3 (2001)*, 109-120.