슬라이딩 윈도에서의 데이터 스팀데이터 유사 질의 처리를 위한 다중질의 최적화 기법*

이양파*, 이연*, 신승선*, 이동욱*, 정원일**, 배해영*
*인하대학교 정보공학과
**호서대학교 정보보호학과
liliangbohost@gmail.com, {leeyeon, hermit, dwlee}@dblab.inha.ac.kr,
wnchung@hoseo.edu, hybae@inha.ac.kr

A Multi-Query Optimizing Method for Data Stream Similar Queries on Sliding Window

Liangbo Li*, Yan Li*, Song-Sun Shin*, Dong-Wook Lee*, Weon-Il Chung**, Hae-Young Bae*
* Dept. of Computer Science and Information Engineering, Inha University
** Dept. of Information Security Engineering, Hoseo University

Abstract

In the presence of multiple continuous queries, multi-query optimizing is a new challenge to process multiple stream data in real-time. So, in this paper, we proposed an approach to optimize multi-query of sliding window on network traffic data streams and do some comparisons to traditional queries without optimizing. We also detail some method of scheduling on different data streams, while different scheduling made different results. We test the results on variety of multi-query processing schedule, and proofed the proposed method is effectively optimized the data stream similar multi-queries.

1. Introduction

With the development of technology and society, data stream management system become more and more significant along with multi-stream and multi-query. Consider a network traffic management system, the input stream may come from several different sources which generate massive, rapid and real-time information, also there may are many output streams which are very similar but only less requests are different. In the presence of multiple continuous queries, there is clearly an opportunity to share common intermediate data and thus, increase the overall processing speed of the system. It is also a challenge that how to optimizing multi-query effectively.

Traditional databases store sets of relatively static records with no pre-defined notion of time, unless timestamp attributes are explicitly added. While this model adequately represents commercial catalogues or repositories of personal information, many current and emerging applications require support for online analysis of rapidly changing data streams. Limitations of traditional DBMS in supporting streaming applications have been recognized, prompting research to augment existing technologies and build new systems to manage streaming data. A data stream is a sequence of real-time, ordered, continuous items.

Data stream management system process on-line data such as sensor measurements, IP packet headers, stock quotes, and transaction logs. Network traffic monitoring is a compelling application of data stream management system. For instance, the Gigascope DSMS has been developed at AT&T Labs [2]. Applications of Gigascope include traffic analysis, performance monitoring, troubleshooting, and detection of network attacks. Usually, only a sliding window of recently arrived data is available at a given time to avoid memory overflow and to emphasize recent data be useful.

For some special aims, joining and optimizing some multi-inputs using temporal join conditions over temporal windows is important, and also joining and optimizing persistent queries has the same importance, especially persistent queries have similar semantics and be updated periodically with some user-specified frequency or re-execution interval. For example, many queries may compute the same aggregate function on the same attribute, but over different window length and different frequencies. Therefore, multi-query optimization is particularly important.

This paper focuses some issues about optimizing multi-query which have the similar requests and use the same streams. We emphasize the methods of sliding window and

* This research was supported by a grant(07KLSGC05) from Cutting-edge Urban Development - Korean Land Spatialization Research Project funded by Ministry of Construction & Transportation of Korean government.

query scheduling on this paper. The remainders content of this paper are organized as follows: In section 2, we introduce some related work and propose an example to analysis. Section 3 optimizes the queries by using sliding window. Section 4 schedules the queries to improve the performance efficiently. And section 6 concludes this paper.

2. Related Work

Paper named Optimization of Multiple Continuous Queries over Streaming Satellite Data [5] describes a system that realizes multiple query processing using two major components: a query optimizer generates an execution plan specific to the active queries, and a query executor then rewrites this plan into a set of processing steps and executes the plan. Here we do some else work related optimizing to achieve the same aims.

As an example, let us consider the following requests, in which the queries compute aggregates under some packet stream. Representative examples include:

1. For every 1-mimute interval, report the number of DNS requests in that interval (i.e., packets whose destination port equals 80) that do not have a matching response packet from the server in recent 30 seconds (i.e., one whose source port is 80, and destination IP address and port are equal to the source IP address and port of the request).

2. For every 1-nimute interval, report the number of DNS requests in that interval that do not have a matching response packet from the server in recent 50 seconds.

3. For every 1-minute interval, report the number of DNS requests in that interval than do not have a matching response packet from the serve in recent 70 seconds.

4. For every 3-minute interval, report the number of DNS requests in that interval than do not have a matching response packet from the serve in recent 100 seconds.
5. For every 3-minute interval, report the number of DNS requests in that interval than do not have a matching response packet from the serve in recent 150 seconds.

The above queries may be expressed as joins on the IP address, port, and timestamp, followed by computation of aggregate over temporal windows. In this paper, we do not talk about the details of how to implement the query of computing the number of DNS requests satisfied special conditions, because there is paper [3] which has implemented and optimized it. Instead we focus on the difference of these examples and do some research we are interesting on it.

3. Query Evaluation and Optimization

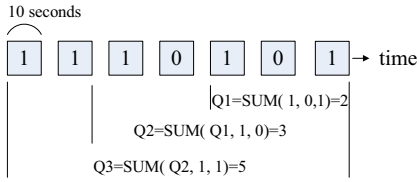
For the previous examples, we define some constraints to simplify the results and to analysis conveniently. Stream S denotes an intermediate stream which joins the multi-stream or sub-stream. Assume that the number of DNS requests which satisfy the special conditions are computed by summarize the attribute of S.

This definition is reasonable. Consider that in a network traffic management system, when there are abundant client streams connecting to the server, computing the number of streams which did not get a matching response during the time period is significant. To achieve this goal, it is necessary to create some attributes to record this important information.

Here, an attribute called Rsp was created, while receiving matching response denoted by 0 others by 1. According to these constraints we can change the previous requests to the following queries:

```
Q1 SELECT SUM(Rsp) FROM
                S [WINDOW 30 sec SLIDE 1min]
Q2 SELECT SUM(Rsp) FROM
                S [WINDOW 50 sec SLIDE 1min]
Q3 SELECT SUM(Rsp) FROM
                S [WINDOW 70 sec SLIDE 1min]
Q4 SELECT SUM(Rsp) FROM
                S [WINDOW 100 sec SLIDE 3 min]
Q5 SELECT SUM(Rsp) FROM
                S [WINDOW 150 sec SLIDE 3 min]
```

The traditional method to deal with this problem is to execute every SQL individual, while may not be very effective. Suppose that Q1, Q2, and Q3 are all due for re-evaluation. They can be answered using a sliding window synopsis illustrated in Figure 1.



(Figure 1: Shared evaluation of SUM aggregate)

The window is partitioned into non-overlapping intervals of 10 seconds each and each interval stores only its sum value. Every ten second, the synopsis is updated by deleting the oldest interval and appending the sum value that has arrived within the last ten second.

To obtain the sum value over a sliding window of 10 *n seconds, we take the sum of the attribute which val ue is 1 of the first n intervals. As illustrated, the answe rs of Q1 and Q2 may effectively be computed for free during the computation Q3---we stop after reading the fi rst three intervals and return an answer of Q1, then rea d the next two intervals and return an answer of Q2, a nd then read the remaining two intervals in order to ans wer Q3.

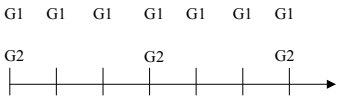
In general, we assume the existence of a set of rules specifying which queries may be efficiently executed together if their re-execution times happen to coincide. In the above example, all five queries may share computation, provided that the synopsis from Figure 1 contains 15 ten-

second intervals to cover the longest window referenced by Q3.

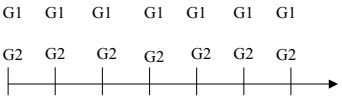
The goal of query optimization is to minimize the processing (or response) time for all active queries in the system. For the types of queries described in the previous section, optimization if primarily concerned the goal: minimizing the overlap time that every query should consume to reduce the total processing time.

4. Query Scheduling Method

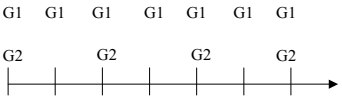
It is obvious that we can improve the query efficiency of the previous example through scheduling the query sequences. Consider a query sequence without any scheduling. Suppose that Q1 to Q5 are partitioned into two groups according to their frequencies: group G1 contains Q1 Q2 and Q3; group G2 contains Q4 and Q5. An execution sequence of Q1 to Q5 is illustrated on a time axis in Figure 2.



(Figure 2: Query sequence without scheduling)



(Figure 3: Query sequence with scheduling)



(Figure 4: Query sequence with scheduling)

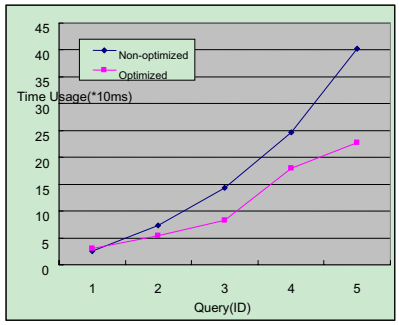
We can compute the costs by computing the SUM aggregation through the longest interval of the synopsis. For example, to G1, there needs 6 SUM to calculate all of the there queries' results, and the cost is 6. By the similar reasoning the cost of G2 is 14. In every 3 minute, queries in G1 are executed separately twice, for a cost of 6*2=12, and both of them are executed together once. Therefore, the total cost of executing queries in G1 and G2 is 12+14=26 per three minutes, or 8.67 per minute.

Other possibility is to synchronize the schedules of all similar queries that may be executed together in order to take advantage of overlapping computation. This means that all five queries above would have to be scheduled every one minute, as illustrated in Figure 3. Another solution is to schedule G2 every two minutes, as illustrated in Figure 4.

Now suppose that G2 is executed whenever G1 is due for a refresh. In this case, the cost per minute of the first scheduling is 14, and the second is 14/2=7. Different scheduling makes different results, which may be effective or ineffective.

5. Performance Experiment

As discussed above, optimized queries are executed in the order, Q1->Q2->Q3>Q4->Q5, the later can share the result of current query. Figure 4 shows that the processing time usage for the optimized queries is much shorter than non-optimized query execution strategy. And with the increasing number of the query requests, the non-optimized strategy curve grows at a more fast speed, which can be summarized from the gap between two curves, which is more and more wide. All above proves the paper idea.



(Figure 5: Cooperation of time usage of multi-queries)

6. Conclusion

For large number of continuous queries, optimizations over multiple queries have been shown to be an effective method to increase the overall system performance. How much savings can be expected depends primarily on the relationships of the queries in the system.

This paper proposed an optimizing method for similar multiple queries on data stream sliding window which could used on network traffic streams. We think it is not a bad choice for optimizing, and also there are many others' methods which maybe better than here's. We are expecting the more effective and efficient technology that can make query optimization greatly.

Reference

[1] Brian Babcock , Shivnath Babu , Mayur Datar , Rajeev Motwani , Jennifer Widom, "Models and issues in data stream systems", Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, June 03-05, 2002, Madison, Wisconsin

[2] Chuck Cranor, Theodore Johnson, Oliver Spataschek, Vladislav Shkapenyuk, "Gigascope: a stream database for network applications", Proceedings of the 2003 ACM SIGMOD international conference on Management of data, June 09-12, 2003, San Diego, California

[3] Lukasz Golab, Theodore Johnson, Nick Koudas, Divesh Srivastava, David Toman, "Optimizing away joins on data streams", March 2008 SSPS '08: Proceedings of the 2nd international workshop on Scalable stream processing system

[4] Lukasz Golab, Kumar Gaurav Bijay, M. Tamer Özsu, " Multi-Query Optimization of Sliding Window

Aggregates by Schedule Synchronization “ November 2006， CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management

[5] Quinn Hart, Michael Gertz, “Optimization of multiple continuous queries over streaming satellite data”, November 2006，GIS '06 : Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems

[6] Walid Aref, Moustafa Hammad, Ann Christine Catlin, Ihab Ilyas, Thanaa Ghanem, Ahmed Elmagarmid, Mirette Marzouk. “Video Query Processing in the VDBMS Testbed for Video Database Research，” November 2003， MMDB '03: Proceedings of the 1st ACM international workshop on Multimedia databases

[7] Brian Babcock , Shivnath Babu , Mayur Datar , Rajeev Motwani , Dilys Thomas, “Operator scheduling in data stream systems,” The VLDB Journal — The International Journal on Very Large Data Bases, v.13 n.4, p.333-353, December 2004

[8] Bin Liu, Amarnath Gupta, Ramesh Jain, “MedSMan: a streaming data management system over live multimedia,” Proceedings of the 13th annual ACM international conference on Multimedia, November 06-11, 2005, Hilton, Singapore

[9] Gautam Das, Dimitrios Gunopulos, Nick Koudas, Nikos Sarkas. “Ad-hoc Top-k Query Answering for Data Streams，” September 2007， VLDB '07: Proceedings of the 33rd international conference on Very large data bases