Shared Execution Techniques for Business Data Analytics over Big Data Streams

Serkan Uzunbaz
Amobee
serkan.uzunbaz@amobee.com

Walid G. Aref
Purdue University
aref@purdue.edu

ABSTRACT

Business Data Analytics require processing of large numbers of data streams and the creation of materialized views in order to provide near real-time answers to user queries. Materializing the view of each query and refreshing it continuously as a separate query execution plan is not efficient and is not scalable. In this paper, we present a global query execution plan to simultaneously support multiple queries, and minimize the number of input scans, operators, and tuples flowing between the operators. We propose shared-execution techniques for creating and maintaining materialized views in support of business data analytics queries. We utilize commonalities in multiple business data analytics queries to support scalable and efficient processing of big data streams. The paper highlights shared execution techniques for select predicates, group, and aggregate calculations. We present how global query execution plans are run in a distributed stream processing system, called INGA which is built on top of Storm.

In INGA, we are able to support online view maintenance of 2500 materialized views using 237 queries by utilizing the shared constructs between the queries. We are able to run all 237 queries using a single global query execution plan tree with depth of 21.

CCS CONCEPTS

• **Information systems** → *Online analytical processing engines*; *Stream management*.

KEYWORDS

big data, stream processing, business data analytics, online view maintenance, INGA

ACM Reference Format:
Serkan Uzunbaz and Walid G. Aref. 2020. Shared Execution Techniques for Business Data Analytics over Big Data Streams. In *32nd International Conference on Scientific and Statistical Database Management (SSDBM 2020), July 7–9, 2020, Vienna, Austria.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3400903.3400932

1 INTRODUCTION

Real-time processing is becoming more and more important for internet companies and their customers. It allows them to identify opportunities as they occur, thus acting upon them in real-time. Companies often receive a lot of their data as continuous big data streams, e.g., search queries, posts and tweets, page views (impressions), ad click streams, etc. Important signals need to be extracted off the data promptly, and be acted upon. For example, in an advertisement campaign scenario, is the new online advertising campaign performing well? Is the company getting a satisfactory volume of clicks from users with the desired profiles? Detecting a discrepancy allows the advertiser to promptly take corrective actions to save money, e.g., by changing the campaign's targeting criteria. Moreover, users need to run online analytical queries against this data (both fresh and historical) to get insights in real-time. For example, consider the following query: "What is the distribution of clicks the ad campaign has been receiving over the last month broken down by day?". This query requires storing this high volume data as well as indexing it to be able to answer these queries in real-time. Having multiple queries of the same flavor provides an opportunity for optimizing their performance.

The use cases and sample queries above are examples of Business Data Analytics queries where large numbers of big data streams need to be processed in real-time by creating and materializing views to continuously provide answers to users' queries.

The views created in Business Data Analytics can be considered as Online Analytical Processing (OLAP) queries with consolidations (roll-ups) based on some conditions (filters). These materialized views are created once, and then are read multiple times for each user query, are refreshed with the arrival of new data, and are updated with modifications in the set of business data analytics queries as depicted in Figure 1.

Not only do these big data streams have events arriving at high rates, but also each event is often information rich carrying high-dimensional data. For example, Amobee receives many TBs worth of events daily. Each event contains hundreds of fields. For instance, an impression event would have the attributes of the page viewed as well as the profile of the user viewing the page in addition to the context, where the impression takes place, e.g., the user's geographical location, the browser's type and version, the device name and type, etc. Data for events are often denormalized. This is to avoid the expensive cost of joins.

To analyze this large-volume of high-dimensional data, users often slice and dice the data in many ways aggregating it over different dimensions. This results in a large number of queries operating over the same data stream. For example at Amobee, we have hundreds queries operating over the same stream. These queries have lots of commonalities and overlap, and thus creating a big efficiency opportunity for shared execution among these queries.

In order to tackle the aforementioned opportunity, we have developed INGA, a real-time big-data stream-processing system. INGA is

SSDBM 2020, July 7–9, 2020, Vienna, Austria

ACM ISBN 978-1-4503-8814-6/20/07…$15.00
https://doi.org/10.1145/3400903.3400932

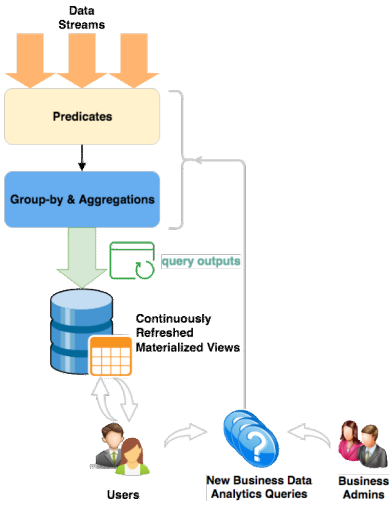


Figure 1: Materialized Views in Business Data Analytics

built to execute thousands of business data analytics queries against the streams, automatically detect overlaps among the concurrent and arriving queries, and produce an optimized shared-execution query evaluation plan that exploits shared execution.

Previous work handles big data streams. In the literature, MapReduce Online extends MapReduce to handle streams. In industry, many systems have been built to handle big data streams, e.g. Storm, MillWheel, Samza, and S4. INGA complements this previous work. It provides a multi-query stream processing engine, includes a query optimizer as well as storage and indexing of the queries' output streams for future retrieval.

In this paper, we highlight the shared execution engine for business analytics queries in INGA. The main contributions of this paper are as follows.

1. We present INGA that provides a multi query optimization engine where logical query execution plans for all the queries are combined into one global query execution plan that shares the execution of the queries' common select predicates, aggregate functions, and group-by dimensions. The query optimizer for INGA uses novel SubsetHeap and SupersetHeap data structures to facilitate the generation of the optimized global query execution plans. INGA operates on a distributed system for scalable real-time computation.

2. We design and prototype INGA, evaluating it using a real ad impressions data stream. The experimental results demonstrate that INGA can support online view maintenance of 2500 materialized views using 237 queries by utilizing the shared constructs, and decrease the end-to-end latency into the order of few seconds using a single global query execution plan tree with depth of 21.

2 BACKGROUND

An examination of typical business data analytics queries shows that these queries create views on the same type of data streams that are maintained continuously as the data arrives, and are mostly filtered by common predicates, have group-by on common keys, contain common aggregates, and join the common data streams using common join keys. These commonalities in business data analytics queries give us the opportunity to combine multiple of these queries during query execution for scalability and efficiency. Shared execution of the common operations in these continuous queries is vital because the underlying input data streams these queries operate on have high arrival rates.

There are two main challenges when processing continuous business data analytics queries that create materialized views that then serve as the source for answering user queries. The first challenge is: *How to minimize the number of times the input stream is read.* Reading a specific input stream with high input rate for each query is not scalable. For example, reading the input stream of impressions for each advertiser query is not efficient. The second challenge is: *How to minimize the number of operators applied and the number of tuples flowing between these operators.* Identifying the shared operators among the running queries, processing these shared operators once, and sharing their output tuples among the downstream operators makes query processing more efficient and more scalable.

Running Example 1:
Consider the following real-world queries from online advertising business analytics. The views created by these sample queries are endpoint pages or reports that are hit and are queried by real users. In this example, the intended users are the advertisers.

q_1:
```
SELECT advertiser_id, insertion_order_id,
SUM(impressions), SUM(cost), SUM(bid_price)
FROM impressions
WHERE country = 'US'
GROUP BY advertiser_id, insertion_order_id
WINDOW 5m
```

q_2:
```
SELECT advertiser_id,
SUM(impressions), SUM(cost), SUM(bid_price)
FROM impressions
WHERE country = 'US' AND state = 'NY'
GROUP BY advertiser_id
WINDOW 5m
```

q_3:
```
SELECT advertiser_id,
SUM(impressions), SUM(cost), SUM(bid_price), SUM(bids)
FROM impressions
WHERE country = 'US'
GROUP BY advertiser_id
WINDOW 5m
```

These queries can be executed in complete isolation, where we can have one query execution plan per query, or they can be executed in a single global query execution plan (Figure 2). In the global query execution plan, the input data stream is read just once instead of three times, as in the separate execution plans.

3 THE DESIGN OF INGA

Shared execution can be achieved by processing as many queries as possible in a single query execution plan. Query execution in INGA is optimized using a global query execution plan where all queries on an input data stream are processed together. The goal is to minimize the number of times we read the tuples from the input data stream, the number of times we read the intermediate tuples in the query evaluation plan, and the total number of tuples flowing between the operators. As a result, both the IO and network load

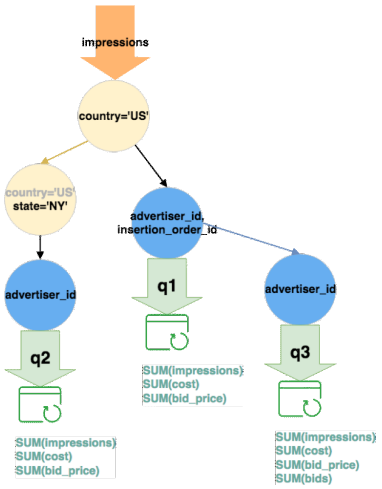
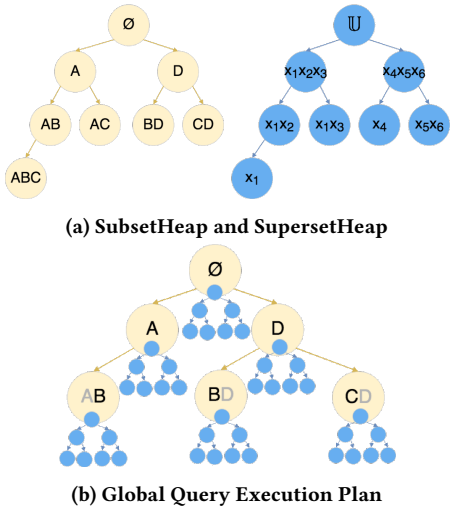


Figure 2: Shared Execution



(a) SubsetHeap and SupersetHeap

(b) Global Query Execution Plan

Figure 3: Proposed heaps and global query execution plan. A, B, C, D are predicates, x_i represents a group-by dimension

operations are minimized, and that leads to an efficient and scalable query execution.

In INGA, we detect commonalities in query constructs, and share the execution of these constructs in order to optimize the total cost of query execution. The detected query constructs are mainly the boolean predicates in `WHERE` clauses, the aggregates in `SELECT` clauses and the `GROUP BY` clauses that accompany these aggregates.

3.1 Global Query Execution Plan

A global query execution plan on one or more input data streams is represented as a heap data structure, where operators are denoted as nodes. For illustration purposes, assume that we are given a set of queries on a single input data stream with select predicates and group-by/aggregate calculations. The global query execution plan represents the select predicates as *predicate nodes* in a *SubsetHeap* and the group-by/aggregate calculations as *group-by nodes* in a *SupersetHeap*. Sample heaps are given in Figure 3a. We define these proposed data structures in the following way:

SubsetHeap is a tree-based data structure (similar to a MinHeap), where

$$A = parent(B) \Rightarrow key(A) \subset key(B).$$

The root node of a *SubsetHeap* is the empty set ($\emptyset$).

SupersetHeap is a tree-based data structure (similar to a MaxHeap), where

$$A = parent(B) \Rightarrow key(A) \supset key(B).$$

The root node of a *SupersetHeap* is the universal set (U).

Then, a global query plan is defined as a SubsetHeap, where there is a SupersetHeap connected to each predicate node if there are queries with predicates that are same as the predicates of that node, as depicted in Figure 3b. The global query execution plan for the sample real-world queries given in Section 2 is illustrated in Figure 2.

4 THE ARCHITECTURE OF INGA

The architecture of INGA is illustrated in Figure 4. Input queries over data streams are processed by the IQL Parser (IQL is INGA's Query Language) that creates the query constructs (predicate and group-by nodes). The validation of the query is checked with the help of the schema files written as proto files (Google Protocol Buffers format) and are given to the ProtoParser for creation of the schemas. Query constructs are then given to Logical Query Optimizer which inserts those into heap data structures (SubsetHeap and SupersetHeap) and outputs a single global query execution plan per input data stream.

The shared execution techniques provided in section 3 are applied in logical query executor in order to create an optimized query execution plan. These global query execution plans (one plan per data stream) are then passed to Physical Query Optimizer which creates execution tasks and distributes the tasks for physical execution at the servers of underlying data stream processing system, Storm. Input tuples arrive at the Storm topologies, get processed and resulting output tuples are written to HBase. The tuples that are stored in HBase form the materialized views which are then queried by end users.

5 RELATED WORK

Our work is related to a broad area of existing works especially in the following two main areas: materialized views and view maintenance, and multi query optimization and shared execution. The works in these main areas can be categorized into two depending on the system in the focus of the research: disk-based traditional relational database management systems and data stream management systems. Therefore, we classify the related work into the following categories: (1) disk-based view maintenance and shared execution, (2) streaming view maintenance and shared execution.

A lot of work has been conducted on view maintenance and multi query optimization or shared execution in the context of traditional relational databases, e.g., [4, 5, 8, 9, 17, 19], and in the context of data stream management systems, e.g., [1–3, 6, 7, 10–16, 18, 20].

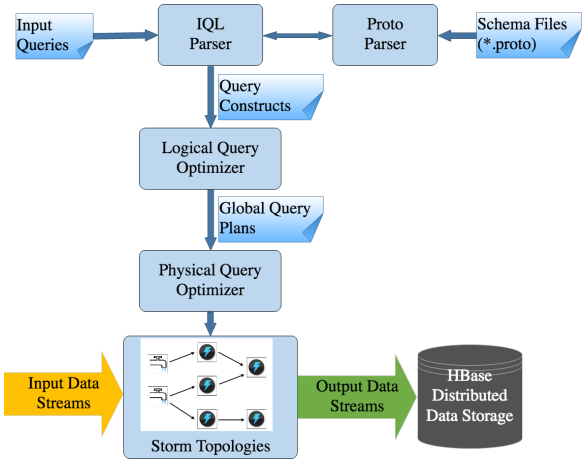


Figure 4: System architecture

The closest work to our proposal is [4], which focuses on sharing the group-by operators in a disk-based relational database. However, it lacks any mention on predicate sharing and application to streaming data is not mentioned.

Many works in the streaming category focus on incremental view maintenance, and shared execution in these works is generally defined as the re-use of these views. We cover the use case of processing business data analytics queries on big data streams while doing operator and predicate level execution sharing and materializing the periodic outputs as views to support user queries. Our work is also novel since it focuses on how global query execution plan can be run a modern distributed processing system.

6 CONCLUSION

We showed that by applying shared execution techniques over the queries of big data streams we can have efficient global query execution plans. These plans can be translated into physical query execution tasks that run on a distributed stream processing system.

Further improvements are possible on the efficiency of global query execution plan. For example, if we consider the predicate containment (e.g., $age > 30$ contains $age > 50$) or group-by dimensions containment (e.g., $GROUP\ BY\ city$ contains $GROUP\ BY\ city, state$) we can detect the shared constructs in the queries even if they do not match.

ACKNOWLEDGMENTS

Walid Aref acknowledges the support of the National Science Foundation under Grant Numbers IIS-1910216 and III-1815796.

REFERENCES

[1] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. 2008. Efficient Pattern Matching over Event Streams. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (Vancouver, Canada) *(SIGMOD '08)*. ACM, New York, NY, USA, 147–160. https://doi.org/10.1145/1376616.1376634

[2] Andreas Behrend, Ulrike Griefahn, Hannes Voigt, and Philip Schmiegelt. 2015. Optimizing Continuous Queries Using Update Propagation with Varying Granularities. In *Proceedings of the 27th International Conference on Scientific and Statistical Database Management* (La Jolla, California) *(SSDBM '15)*. ACM, New York, NY, USA, Article 14, 12 pages. https://doi.org/10.1145/2791347.2791368

[3] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. 2000. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (Dallas, Texas, USA) *(SIGMOD '00)*. ACM, New York, NY, USA, 379–390. https://doi.org/10.1145/342009.335432

[4] Zhimin Chen and Vivek Narasayya. 2005. Efficient Computation of Multiple Group by Queries. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data* (Baltimore, Maryland) *(SIGMOD '05)*. ACM, New York, NY, USA, 263–274. https://doi.org/10.1145/1066157.1066188

[5] Rada Chirkova, Chen Li, and Jia Li. 2006. Answering queries using materialized views with minimum size. *The VLDB Journal* 15, 3 (01 Sep 2006), 191–210. https://doi.org/10.1007/s00778-005-0162-8

[6] Françoise Fabret, H. Arno Jacobsen, François Llirbat, João Pereira, Kenneth A. Ross, and Dennis Shasha. 2001. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data* (Santa Barbara, California, USA) *(SIGMOD '01)*. ACM, New York, NY, USA, 115–126. https://doi.org/10.1145/375663.375677

[7] Thanaa M. Ghanem, Ahmed K. Elmagarmid, Per-Åke Larson, and Walid G. Aref. 2008. Supporting Views in Data Stream Management Systems. *ACM Trans. Database Syst.* 35, 1, Article 1 (Feb. 2008), 47 pages. https://doi.org/10.1145/1670243.1670244

[8] Ashish Gupta and Inderpal Singh Mumick. 1999. Materialized Views. MIT Press, Cambridge, MA, USA, Chapter Maintenance of Materialized Views: Problems, Techniques, and Applications, 145–157. http://dl.acm.org/citation.cfm?id=310709.310737

[9] Himanshu Gupta and Inderpal Singh Mumick. 2005. Selection of Views to Materialize in a Data Warehouse. *IEEE Trans. on Knowl. and Data Eng.* 17, 1 (Jan. 2005), 24–43. https://doi.org/10.1109/TKDE.2005.16

[10] Moustafa A. Hammad, Michael J. Franklin, Walid G. Aref, and Ahmed K. Elmagarmid. 2003. Scheduling for Shared Window Joins over Data Streams. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29* (Berlin, Germany) *(VLDB '03)*. VLDB Endowment, 297–308. http://dl.acm.org/citation.cfm?id=1315451.1315478

[11] Manhui Han, Jonghem Youn, and Sang-goo Lee. 2017. Efficient Query Processing on Distributed Stream Processing Engine. In *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication* (Beppu, Japan) *(IMCOM '17)*. ACM, New York, NY, USA, Article 29, 8 pages. https://doi.org/10.1145/3022227.3022255

[12] Mingsheng Hong, Mirek Riedewald, Christoph Koch, Johannes Gehrke, and Alan Demers. 2009. Rule-based Multi-query Optimization. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology* (Saint Petersburg, Russia) *(EDBT '09)*. ACM, New York, NY, USA, 120–131. https://doi.org/10.1145/1516360.1516376

[13] Ryan Huebsch, Minos Garofalakis, Joseph M. Hellerstein, and Ion Stoica. 2007. Sharing Aggregate Computation for Distributed Queries. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data* (Beijing, China) *(SIGMOD '07)*. ACM, New York, NY, USA, 485–496. https://doi.org/10.1145/1247480.1247535

[14] Albert Jonathan, Abhishek Chandra, and Jon Weissman. 2018. Multi-Query Optimization in Wide-Area Streaming Analytics. In *Proceedings of the ACM Symposium on Cloud Computing* (Carlsbad, CA, USA) *(SoCC '18)*. ACM, New York, NY, USA, 412–425. https://doi.org/10.1145/3267809.3267842

[15] Sailesh Krishnamurthy, Michael J. Franklin, Joseph M. Hellerstein, and Garrett Jacobson. 2004. The Case for Precision Sharing. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30* (Toronto, Canada) *(VLDB '04)*. VLDB Endowment, 972–984. http://dl.acm.org/citation.cfm?id=1316689.1316773

[16] Sailesh Krishnamurthy, Chung Wu, and Michael Franklin. 2006. On-the-fly Sharing for Streamed Aggregation. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data* (Chicago, IL, USA) *(SIGMOD '06)*. ACM, New York, NY, USA, 623–634. https://doi.org/10.1145/1142473.1142543

[17] Shaosu Liu, Bin Song, Sriharsha Gangam, Lawrence Lo, and Khaled Elmeleegy. 2016. Kodiak: Leveraging Materialized Views for Very Low-latency Analytics over High-dimensional Web-scale Data. *Proc. VLDB Endow.* 9, 13 (Sept. 2016), 1269–1280. https://doi.org/10.14778/3007263.3007266

[18] Samuel Madden, Mehul Shah, Joseph M. Hellerstein, and Vijayshankar Raman. 2002. Continuously Adaptive Continuous Queries over Streams. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data* (Madison, Wisconsin) *(SIGMOD '02)*. ACM, New York, NY, USA, 49–60. https://doi.org/10.1145/564691.564698

[19] Timos K. Sellis. 1988. Multiple-query Optimization. *ACM Trans. Database Syst.* 13, 1 (March 1988), 23–52. https://doi.org/10.1145/42201.42203

[20] Yuke Yang, Lukasz Golab, and M. Tamer Özsu. 2017. ViewDF: Declarative Incremental View Maintenance for Streaming Data. *Information Systems* 71 (07 2017). https://doi.org/10.1016/j.is.2017.07.002