Development of an algorithm for generating queries for a data flow control system

I D Kotilevets, N V Ksenofontov, E S Karbova and I A Ivanova

KB-14 Department «Digital data processing technologies», MIREA – Russian
Technological University. 78 Vernadsky Avenue, Russia, Moscow, 119454

E-mail: kotilevec@mirea.ru

Abstract. Nowadays, the process of technology development can be described as accumulation of huge data resources, which numbers are growing. Problems associated with the presentation of information in the form of data flow or streams arise more and more often due to the increased amount of data and the speed of their processing. This article presents inquiry algorithm, which is based on methods of inquiry adaptation. The algorithm includes streaming data processing from different sources, which allows to apply new inquiries to old data and new data to old inquiries to make getting the results more efficient.

1. Introduction
Currently, a large amount of data comes from different sources. In turn, all these data are various models, for example, a relational model, an XML data model, an object-relational model, an ontological data model, and others. To solve various problems, access to various information resources is often required, which requires the formation of a special infrastructure.

There are also more and more resources that provide large flow or streams of real-time data. In addition, there are also tasks that require processing both streaming data and integrated access to these resources.

Recently, the most widespread class of problems, the information in which is presented in the form of data streams. Examples of such tasks are environmental monitoring, financial applications, network monitoring, security tasks, telecommunications data management, network applications, scientific tasks, etc.

At the moment, there are several technologies for processing data streams or flow in real time. Also, traditional technologies, such as database management systems, are able to maintain data in main memory, and rule processors can be used to solve the above tasks. However, none of the out-of-the-box solutions provide the ability to integrate streaming data. This is very important when it comes to solving complex problems, when data comes from more than one source. The main goal of this study is to develop an algorithm for generating queries to streaming databases.

2. Streaming data and the complexity of its processing
Streaming data is data generated continuously by thousands of sources that typically send data records simultaneously and in small volumes (at the level of several kilobytes). Data streaming involves real-time processing of data from thousands of sources such as sensors, financial marketplace transactions, e-commerce purchases, web and mobile apps, social media, and more [7].
This data must be processed sequentially and incrementally for each of the records. Also, the data can be processed using a sliding time window, after which it can be used in various analytical tasks, including correlation, aggregation, filtering and templating [1][2].

The flowing data model differs from the classical relational model in that in the streaming data model, all data or a certain part of it is not available for retrieval from memory, as it comes as one or more continuous data streams.

The main differences from the relational model are the following:

- New data is streamed in real time;
- The system doesn’t control the order in which the data arrives;
- Data flows are not limited in size;
- After an element from the data stream has been processed, it is discarded and no longer taken into account or archived. Such data cannot be stored due to limited memory space.

Stream or flow processing is similar to other types of data processing: a program code (system) is written that receives data, transforms and groups them, and displays the results [3].

A data flow control system is a distributed in-memory data flow control system. It is designed to use SQL queries that are standards compliant to process unstructured and structured data streams in real time.

An important feature of data flow control systems is the ability to handle potentially infinite and rapidly changing data streams. This implies flexible processing, despite the fact that there are only limited resources, such as main memory.

One of the main problems in any streaming processing system is the need to constantly maintain the ability to receive and store messages in real time, especially with large amounts of data. Processing must be done in such a way as to avoid blocking the data ingestion pipeline. It is also necessary to use data stores that support write operations in large volumes.

Another challenge is creating rapid response capabilities such as real-time alerts or real-time (or near real-time) dashboard views [4].

3. Development of an algorithm for generating queries

In new applications, special queries also require processing of data that comes before the query is sent or during the outage period.

This article proposes an algorithm that combines the processing of ad hoc and continuous queries by symmetrical processing of data and queries. This will allow new queries to be applied to old data and new data to old queries.

This algorithm also supports discontinuous connectivity by separating the process of computing query results from the process of delivering those results. The system is based on adaptive query processing methods [5].

There are two scenarios:

Data recharging is the process by which devices periodically connect to the network to update their data content.

Monitoring. A user wants to track some statistical information, such as the number of downloads of music files from their subnet in the last hour, or, for example, the latest posts on a social network with a rating above a certain value.

To support such applications, the designed algorithm for the query generation system or simply the query processor is intended.

The basic idea is that data and queries are streaming. It is also important that data and queries duplicate each other: the processing of multiple queries is considered as the connection of the query and data streams. In addition, the developed algorithm also partially materializes the results to support offline operation, as well as to improve data throughput and query response time.

A data stream management system (DSMS) is a distributed in-memory data flow control system designed to use SQL queries to process real-time memory data flows. Unlike conventional DBMS, in which SQL queries, when executed, return a result and terminate, the same queries executed in a DBMS
do not complete their execution, but continuously generate results as new data arrives. Continuous SQL queries in DBMS use SQL Window functions to analyze, connect, and aggregate data streams over fixed or sliding time windows [6].

So, the user interacts with the system, having previously formed the query structure. The system returns a handle to the user, which can then be reused to later invoke the query results. The user can also cancel the previously specified query.

As described earlier, the client starts by registering a query specification with the system. Query specifications look like:

```
SELECT select_list
FROM from_list
WHERE conjoined_boolean_factors
BEGIN begin time
END end_time
```

The system assigns a unique identifier to the new query (queryID field), which is returned to the user as a determinant for future calls.

When a new tuple of data enters the system, it is assigned a unique identifier and a physical time stamp that matches the system clock. Next, the tuple is embedded in the desired data module (each stream always has one module). It is then used to test the query module to determine which SQC it matches.
The resulting data tuples can be used to further explore other modules in terms of evaluating connection queries. Unique and physical identifiers are stored in the result structure after validation.

**Figure 1.** Processing a selection query: entering a new query

Figure 1 shows the system's actions during the receipt of a new set of SQCs.

Figure 1(a) shows the state of the query module and data module after the system has processed queries with identifiers up to and including 23 and sets of data fields up to and including 52.

Next, we consider the input of a new SQC into the system, this is shown in Figure 1 (b).

That query is assigned an ID of 24 and is inserted into the appropriate module by adding an entry as a pair of identifier and condition - query predicate (queryID, QueryPredicate).

The query results structure is complemented with a new column (Figure 1(b)) to store the query results from the conditions of the received SQC.

Then the query is sent to the data module check, where it is matched against each row of data (Figure 1(c)).
When there are data rows (tuples) that satisfy the incoming query (in the current example, these are rows with identifiers 48 and 50), the system marks these records in the result structure as TRUE (Figure 1 (e)). Analogically, when a new dataset arrives, it is added to the data module, and then, using the query module, it is compared with all current SQCs in the system to update the result structure.

4. Combining queries across multiple threads

If a query contains several data-streams the processing we have seen before is used.

Processing multiple data-streams means join query-stream with all the data streaming from the FROM query list. A symmetric connection is to be generalized to accept more than two input streams.

The system processes a query on two data streams: R & S. Figure 3 illustrates actions performed by the system when a new query enters it. Datasets from R & S have already been proceeded to ID54, as well as queries with identifiers up to 22. Each data-stream has its own data module. One data query module is used for the query stream. Data query modules contain specified data and queries, respectively.
Step 1. System receives query with ID 23. Query contains two factors as for checking: R, S. In this case factors are called data modules, containing some variables.

   In this case R.a < 5, S.b > 1 and R.a > S.b.

Step 2. Query attaches itself to existing queries in query module and it’s used to check modules R and S. Let’s imagine, that query will firstly check data module R.

Step 3. System checks every row in data module to be equal to condition row from query. In the beginning only fields, that fully satisfy queries for R. Data for R is being added to binding thread modules that bind two threads. After this, S data is being checked to be equal to query conditions.

Step 4. Hybrid structure is being formed from fields, that satisfy query conditions. This structure will contain partly calculated query conditions for every R strings.

Step 5. Created hybrid structure will be used to check S data module. For each set of S fields, that satisfy logical factors of entering query, structure of results will be updated the following way: note for pair is being created (R-identifier, S-identifier) and inserted in structure of results of this pair.

Step 6. Received record gets mark, that it satisfies query with ID 23.

On next scheme (figure 4), new record is being put in R module, after what checking is started in query system. Next part of treatment is as on previous steps.

It’s needed to point, that hybrid structures contain some kind of excess (showed on shaded parts of figures). New pack of SQC repeats in hybrid structures as seen in figure 3, as well new data set repeats in figure 4. That leads to repeated treatment in step 5.

5. Calling the query and building the result
The result structure contains information on sets of fields in the data modules, which match the SQC conditions in the query module. For each result of each query, two kinds of identifiers are stored: unique and physical. The physical identifier is formed from the timestamp, while a unique identifier is an
identifier of the query itself, which access the results. Physical identifier is used to sort and index the results: it allows to retrieve data effectively within a time window.

The time window data (taken from the BEGIN-END fields) is retrieved from the WindowsTable and the current endpoint values of the input window are determined, when the user queries the current query result. Since all the data has been already merged using the SQC query specification due to background processing, the required results are already present in the corresponding structure.

Thus, the system engine directly accesses this structure and applies the time window of current query input to its contents to obtain the IDs of the base sets, which form the result sets. Then they can be retrieved from the data module using unique identifiers and returned to the client.

There is no difficulty in getting the current window from the timestamp of the result tuples for queries with a single processing thread. This process is more complicated for connection queries because their results consist of multiple base tuples, each with its own timestamp.

Experimental verification of this algorithm was carried out using the following set of software tools: Apache Storm as the core of the query system, Apache Kafka as a query broker, and Twitter API as a source of streaming data. Data was sampled, including several queries at the same time. The algorithm was implemented in Java and Python. The resulting combination of the algorithm and auxiliary tools showed a high speed of query processing and data acquisition.

On Figure 5 shows an example of query execution in the test system.

![Figure 5](image-url) 

**Figure 5.** An example of query execution in a test system.

6. Conclusion
In this article, the algorithm for generating queries was considered in detail. The algorithm is based on adaptive query processing methods. An algorithm for the formation of queries has been compiled, in relation to the developed software.

Describes the result structure and processing performed by the system to return query results when a previously specified query is invoked.

References
[1] Data in Flight. [https://queue.acm.org/detail.cfm?id=1667562](https://queue.acm.org/detail.cfm?id=1667562)
[2] What is streaming data. [https://aws.amazon.com/streaming-data/](https://aws.amazon.com/streaming-data/)
[3] Narhid N, Shapira G and Palino T 2017 *Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale* (Sebastopol: O'Reilly Media) p 320
[4] Ilyushechkin, V.M. 2019 *Basics of using and designing databases* (Moscow: Urait) p 214
[5] Babu S and Bizarro P 2005 Adaptive Query Processing in the Looking Glass 2nd Biennial Conf. on Innovative Data Systems Research (Asilomar, USA, 4-5 January 2005) p 12

[6] Cugola G and Margara A 2012 Processing flows of information: From data stream to complex event processing. ACM Computing Surveys 44 15

[7] Tomashevskaya V S and Yakovlev D A 2021 Research of unstructured data interpretation problems. Russian Technological Journal. 9 7-17