論文 / 著書情報
Article / Book Information

Title	Dynamic Modification of Continuous Queries by Using RDF Metadata of Information Sources
Author	Yousuke Watanabe, Haruo Yokota
Journal/Book name	Proc. of 015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), , , pp. 754-759
Issue date	2015, 11
DOI	http://dx.doi.org/10.1109/3PGCIC.2015.105
URL	http://www.ieee.org/index.html
Copyright	(c)2015 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other users, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works for resale or redistribution to servers or lists, or reuse of any copyrighted components of this work in other works.
Note	このファイルは著者（最終）版です。 This file is author (final) version.

Dynamic Modification of Continuous Queries by Using RDF Metadata of Information Sources

Yousuke Watanabe
Institute of Innovation for Future Society
Nagoya University
Furocho, Chikusa-ku, Nagoya, Japan
watanabe@coi.nagoya-u.ac.jp

Haruo Yokota
Graduate School of Information Science and Engineering
Tokyo Institute of Technology
2–12–1 Oh-okayama, Meguro, Tokyo, Japan
yokota@cs.titech.ac.jp

Abstract—**Continuous queries over data streams have been used in many applications. They incrementally produce their query results according to data arrival and time progress. However, existing continuous query languages strongly assume that all information sources to be accessed by queries are explicitly specified and permanently fixed. There are more sophisticated requirements such as changing information sources and rewriting filter conditions during query execution. This paper proposes a dynamic query modification scheme for continuous queries. Our continuous query language treats not only stream data from information sources, but also metadata of the information sources and queries themselves. Internally, a continuous query is represented as a dataflow, which is an RDF graph of operators. Thus, modifying a continuous query can be regarded as rewriting the corresponding RDF graph. Searching information sources which satisfy the conditions can be also regarded as searching RDF graphs descripting metadata of information sources. Our scheme achieves both continuous searching information sources and rewriting dataflows.**

Keywords*—*Data stream; Continuous query; RDF; Dynamic rewriting;

I. Introduction

The number of stream-based information sources such as sensors and cameras is increasing. Query processing on stream data has become one of important research issues. Thus, Data Stream Management Systems (DSMSs)[1], [2], [3], [4], [5], [6], [7], [8], [9], which achieve query processing on stream data, have been focused.

The query processing scheme in DSMSs is called continuous query [10], [11]. Since stream data are continuously produced, continuous queries are repeatedly evaluated. According to data arrival and time progress, a DSMS produces query results incrementally. Once a query is posed to a DSMS, its query processing is continued until the owner unregisters it explicitly. Continuous queries have much longer lifetimes than conventional queries in DBMSs.

During execution of continuous queries, the real-world changes dynamically. Statistics of data, arrival rates of data, available computational resources (CPU load), and available network bandwidth may change over time. How to detect changes in the environment and how to adapt DSMSs to the changes have been studied [12], [13], [14], [15], [16], [17].

However, the previous work does not consider that user requirements may also change during query execution. Existing continuous query languages strongly assume that all information sources to be accessed by queries are explicitly specified and permanently fixed. For example, a user would like to get stream data from the nearest information sources according to user's location. This means that we cannot specify an information source to be accessed. The continuous query should be dynamically rewritten based on the user's location. To deal with above situation in existing continuous query languages, we have to interrupt query execution and manually rewrite the query. It is inconvenient when changes occur frequently.

Therefore, we propose flexible query specification scheme that allow dynamic modification of continuous queries. Our continuous query language treats not only stream data from information sources, but also metadata of the information sources and queries themselves. Internally, a continuous query is represented as a dataflow, which is an RDF (Resource Description Framework [18]) graph of operators. Thus, modifying a continuous query can be regarded as rewriting the corresponding RDF graph. We also use RDF graphs to express metadata of information sources. Searching information sources which satisfy the conditions can be also regarded as searching RDF graphs descripting metadata of information sources. Our scheme achieves both continuously searching information sources and rewriting dataflows.

We summarize the contributions of this paper:

1) we point out a limitation of existing continuous query languages.
2) we propose a new query specification scheme with dynamic modification for continuous queries.

The remaining part of this paper is organized as follows: Section II presents an example scenario. Section III describes a data stream management system we assume in this paper. Section IV proposes our query modification scheme. Section VI introduces related work. Section VII concludes this paper.

II. Example Scenario

To illustrate our objective, we show an example scenario. Here, we suppose a kind of moving object tracking (Figure 1). A person to be monitored (the tracking target) freely moves in the two dimensional space. The position of the tracking target is detected by GPS and provided as stream data. Multiple network cameras are located in the space. These cameras

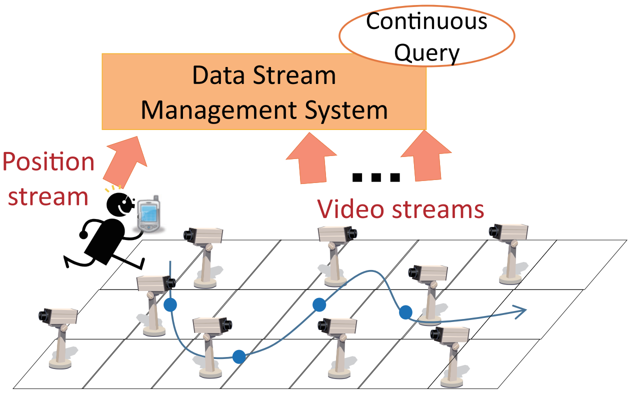


Fig. 1. Example Scenario

provide live video streams via network when we access them. We assume that a location of each camera and its schema are known.

An example requirement in this environment is "a user would like to receive video data from cameras which are located near the tracking target (distance between the tracking target and a camera is less than 10)". Since a camera to be accessed depends on the current position of the tracking target, we cannot specify an information source in advance.

In the existing continuous query languages, there are two approaches to achieve this requirement.

1) *Rewriting queries*: When the current position of the tracking target is notified, we can identify cameras near the tracking target. A new continuous query is generated by using the name of the camera. However, according to movement of the tracking target, we need to interrupt execution of the query and rewrite the query again.
2) *Union of all video streams*: Union operator enables us to assemble one stream including video data from all cameras. However, this approach consumes much computational resources and network bandwidth, because it needs to collect all data from information sources.

The existing continuous query languages are originated from the query languages in DBMS such as SQL. In SQL, there is a strong assumption that a user has enough knowledge about relations to be accessed. A FROM clause in an SQL query should be fixed during query execution. The existing continuous query languages still keep this assumption. This is the cause of the above limitation.

A. Requirement Analysis

To cope with above requirement, a DSMS should perform the following steps.

1) The DSMS obtains position data of the tracking target from GPS stream.
2) The DSMS finds a camera (C_{near}) which is close to the tracking target. It computes distance between the tracking target and camera. We need to provide a position of each camera.
3) Once a camera to be accessed is found, we can construct a dataflow to handle the video stream from the camera. The DSMS establishes a connection to the camera. We need to provide an IP address and an access method (URL) for the camera C_{near}.
4) The DSMS receives video data from camera C_{near}. We need to provide a schema information of $C_{near}(A_1, \ldots, A_n)$. Video data are also processed by continuous queries in the DSMS.
5) The DSMS closes a connection to a camera, when the camera becomes no longer close to the tracking target.

We need to provide camera's metadata such as position, IP address, access method and so on.

III. DSMS Architecture for Dynamic Modification

In this section, we describe an architecture of DSMS in this paper. We assume that data stream provided from information sources are based on relational model. Each data item is represented as a tuple. From Section II-A, metadata handling is also needed. In our proposal, we employ RDF [18] format to represent metadata. We achieve a mechanism which can integrate relational stream and metadata.

A. Architecture

Figure 2 shows an architecture of the DSMS we propose. It consists of the following components:

- Query Parser: A user specifies his requirement as a continuous query. Our query language is explained in Section IV. A continuous query is analyzed by Query parser and converted into a dataflow. A dataflow is a graph of operators. It is registered into both Executor and Graph store.
- Executor: Executor processes stream data. When a new tuple is obtained, Executor applies operators to the tuple according to registered dataflows. In this paper, we consider basic relational operators.
- Wrapper: A wrapper is a component to receive tuples from an information sources. By wrapper, the data format of the information source is converted into relational stream. A type of wrappers should be chosen based on the access method of the information source.
- Wrapper Manager: An instance of wrappers is created/deleted by Wrapper Manager.
- Graph store: Graph store manages RDF graphs. Metadata of information sources and dataflows are stored.

B. Metadata of information sources

Metadata of information sources are represented as RDF graphs. Figure 3 shows an example of metadata descripting information sources. A data item in RDF graph is a triple $(subject, predicate, object)$. In the figure, subjects and objects are expressed by nodes. Predicates are expressed by edges. $Camera1$ is an instance of $InformationSource$. It has two attributes $Camera1.A1$ and $Camera1.A2$. The data type of $Camera1.A2$ is video data.

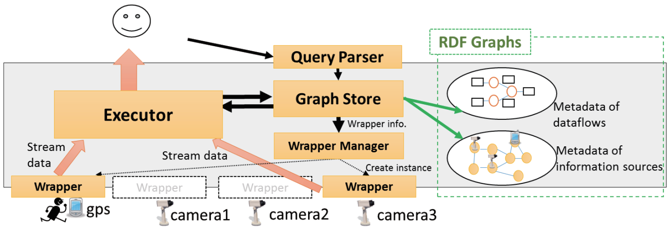


Fig. 2. System architecture

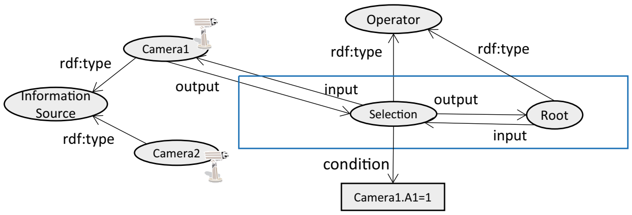


Fig. 4. Metadata descripting dataflow

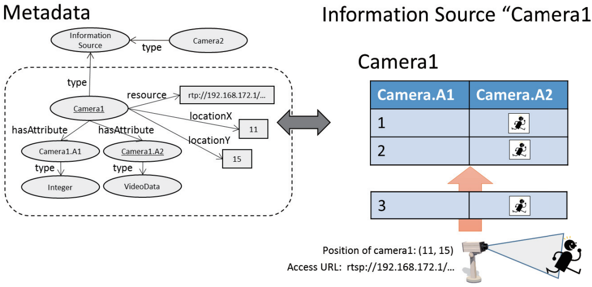


Fig. 3. Metadata descripting information sources

C. Metadata of dataflow

A continuous query written in our query language (Section IV) is converted into a dataflow by Query Parser. A dataflow is a directed-acyclic-graph (DAG) of operators. It is also represented as a RDF graph.

Figure 4 shows an example of metadata descripting a dataflow. This dataflow includes two operators: root and selection. A selection reads tuples from $Camera1$. By rewriting a RDF graph, we can modify the corresponding dataflow. Suppose we would like to access $Camera2$ instead of $Camera1$. In this case, we add an edge from $Camera2$ to the selection, and remove the edge from $Camera1$ to the selection.

IV. Query Specification

Based on the requirement analysis in Section II-A, our query specification scheme needs to provide the following functions (Figure 5):

- Metadata search: In the example, the DSMS finds cameras which are close to the tracking target. Generally, we need to retrieve RDF graphs.
- Handling both stream data and RDF graphs: A DSMS must treat both relational stream and metadata. In the example, the position of the tracking target is converted into a filter condition for metadata of information sources. A dataflow to process video data is constructed from metadata of cameras. We need bidirectional conversion from relational stream to metadata and vice versa.
- Manipulating dataflows in the query language: Since information sources to be accessed change dynamically, we cannot write the name of information source directly. We should provide variables which are dynamically replaced with actual source name and column name.
- Dynamic connection establishment to sources: DSMS has to dynamically establish connections to information sources by using metadata of the information sources.

The proposed query language is based on a hybrid of CQL [11] and SPARQL [19]. CQL-like queries are for stream data, and SPARQL-like queries are for RDF graphs.

Figure 6 is an example query corresponding to the example scenario. SELECT–FROM clauses in Line 4-5 follow the CQL syntax. They specify stream processing for video data from cameras. $?camera$ and $?video$ are metadata variables, whose actual values are continuously obtained by evaluating the following METADATA clause. Expressions in a METADATA clause follow the SPARQL syntax (Line 7-15). They specify how to find information sources which satisfy the conditions. $ns : informationSource$ is a RDF object which corresponds to the class of information sources in the DSMS. $ns : locationX$ and $ns : locationY$ are RDF predicates for position coordinates of the tracking target. A VALUES clause in Line 16 is an extension of SPARQL syntax. This clause controls binding between metadata variables and intermediate result of stream processing. Metadata variables $?tx$ and $?ty$ are replaced with a coordinate (x, y) from GPS stream (specified in Line 17-18).

A. Evaluation of queries

Evaluation of the example query is illustrated in Figure 7. When a new data item arrives from GPS stream, values of $?tx$ and $?ty$ are updated. Then, metadata search is performed. The result of metadata search is bound to variables $?camera$ and $?video$. Changing values of $?camera$ and $?video$ means information sources to be accessed change. The DSMS establishes a connection to the new information sources. After that, the DSMS starts to receive video data and applies operators to the video data.

V. Dynamic Connection Management

The DSMS establishes a connection to an information source according to intermediate result of metadata search. It continuously monitors changes in the result of metadata search. When it detects changes, Wrapper Manager in the DSMS creates a new instance of Wrapper.

As described in Section III-B, an access method for each information source is also included in metadata. By retrieving the access method of the information source, Wrapper Manager

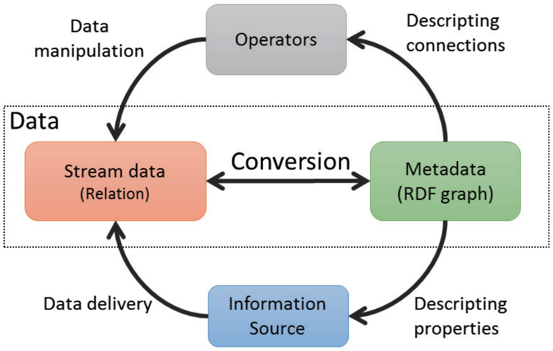
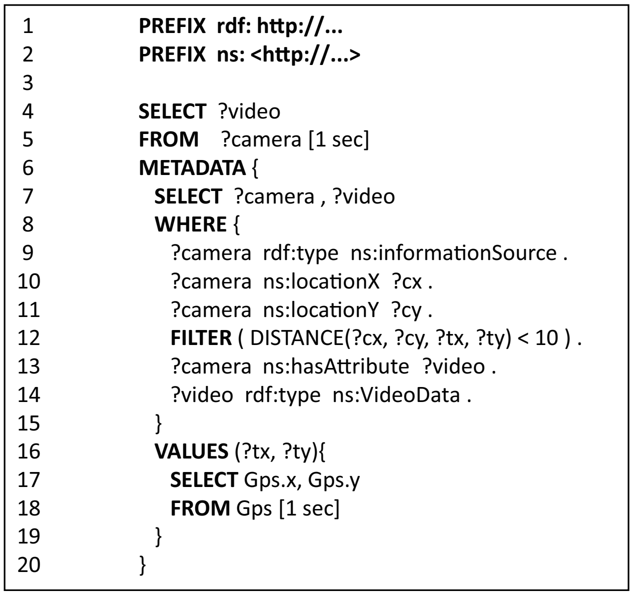


Fig. 5. Relationship between stream data and metadata



```
PREFIX  rdf: http://...
PREFIX  ns: <http://...>

SELECT  ?video
FROM    ?camera [1 sec]
METADATA {
  SELECT  ?camera , ?video
  WHERE {
    ?camera  rdf:type  ns:informationSource .
    ?camera  ns:locationX  ?cx .
    ?camera  ns:locationY  ?cy .
    FILTER ( DISTANCE(?cx, ?cy, ?tx, ?ty) < 10 ) .
    ?camera  ns:hasAttribute  ?video .
    ?video  rdf:type  ns:VideoData .
  }
  VALUES (?tx, ?ty){
    SELECT Gps.x, Gps.y
    FROM Gps [1 sec]
  }
}
```

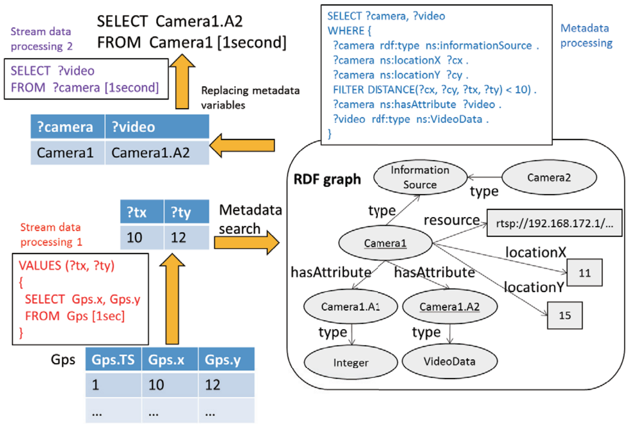
Fig. 6. An example query with dynamic modification

determines the Java class fitting to the information source. An instance of wrapper establishes a connection to the information source.

VI. Related Work

Many Data Stream Management Systems and continuous query languages are proposed in the previous work. The existing continuous query languages are classified into two categories: declarative approach and dataflow approach.

SQL-like declarative query languages are applicable to relational streams (infinite sequence of tuple). CQL [11] is one of major declarative continuous query languages. It provides three types of operators, stream-to-relation, relation-to-relation, relation-to-stream. A sliding-window is a stream-to-relation operator. It gets a finite set of tuples from stream. Relation-to-relation operators are traditional relational operators such as selection and join. I-stream, D-stream, and R-stream are relation-to-stream operators, which are used to obtain differences from the result produced in the previous evaluation. A part of our query language is based on CQL syntax.



```
SELECT  Camera1.A2
FROM  Camera1 [1second]
```

```
SELECT  ?video
FROM  ?camera [1second]
```

```
SELECT ?camera, ?video
WHERE {
 ?camera  rdf:type  ns:informationSource .
 ?camera  ns:locationX  ?cx .
 ?camera  ns:locationY  ?cy .
 FILTER DISTANCE(?cx, ?cy, ?tx, ?ty) < 10) .
 ?camera  ns:hasAttribute  ?video .
 ?video  rdf:type  ns:VideoData .
}
```

?camera	?video
Camera1	Camera1.A2

?tx	?ty
10	12

```
VALUES (?tx, ?ty)
{
 SELECT  Gps.x, Gps.y
 FROM  Gps [1sec]
}
```

Gps.TS	Gps.x	Gps.y
1	10	12
...	...	...

Fig. 7. Evaluation of the example query

C-SPARQL [20] is a declarative continuous query language for RDF streams. Its syntax is based on SPARQL. A data item in C-SPARQL is a pair of timestamp and RDF triple (timestamp, (subject, predicate, object)).

In some DSMSs [4], [5], [6], [7], a user can specify dataflows directly. A dataflow is a DAG of operators handling stream data. Dataflow approach is applicable to not only relational streams but also object streams (infinite sequence of objects). Our query language is converted to dataflow inside of the DSMS.

Existing continuous query languages strongly assume that all information sources to be accessed by queries are explicitly specified and permanently fixed. This paper proposes a dynamic query modification scheme for continuous queries.

VII. Conclusion

This paper pointed out the limitation of existing continuous query languages. Most continuous query languages are originated from query language in database systems. They do not consider that information sources accessed by queries may change dynamically. To avoid the problem, this paper proposed dynamic modification scheme for continuous queries. It can handle both RDF graph and stream data. Its syntax is a hybrid of CQL and SPARQL. By using metadata, our DSMS can establish connections to information sources to be accessed.

We have been developing a prototype system. Performance evaluation is one of future research issues.

Acknowledgment

This work was supported in part by JSPS KAKENHI Grant Number 24700087 and 25240014.

References

[1] J. Chen, D. J. DeWitt, F. Tian, Y. Wang. NiagaraCQ: a scalable continuous query system for Internet databases. Proc. ACM SIGMOD, pp. 379-390, 2000.

[2] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. In Proc. CIDR, pages 245-256, 2003.

[3] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. Telegraphcq: Continuous dataflow processing. In Proc. the 2003 ACM SIGMOD International Conference on Management of Data, pages 668-668. 2003.

[4] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A new model and architecture for data stream management. The VLDB Journal, 12(2):120-139, 2003.

[5] D. J. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J. hyon Hwang, W. Lindner, A. S. Maskey, E. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the borealis stream processing engine. In Proc. CIDR, pp. 277-289, 2005.

[6] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. S4: Distributed stream computing platform. In Proc. the 2010 IEEE International Conference on Data Mining Workshops, pages 170-177, 2010.

[7] Storm project, https://storm.apache.org/

[8] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica. Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters. In Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Ccomputing, HotCloud ’12, pages 10-10, 2012.

[9] U. Cetintemel, J. Du, T. Kraska, S. Madden, D. Maier, J. Meehan, A. Pavlo, M. Stonebraker, E. Sutherland, N. Tatbul, K. Tufte, H. Wang, S. B. Zdonik. S-Store: A Streaming NewSQL System for Big Velocity Applications. In Proc. VLDB, pp. 1633-1636, 2014.

[10] D. Terry, D. Goldberg, D. Nichols, and B. Oki. Continuous queries over append-only databases. SIGMOD Rec., 21(2):321-330, 1992.

[11] A. Arasu, S. Babu, and J. Widom. The cql continuous query language: Semantic foundations and query execution. The VLDB Journal, 15(2):121-142, 2006.

[12] R. Avnur and J. M. Hellerstein. Eddies: continuously adaptive query processing. In Proc. the 2000 ACM SIGMOD international conference on Management of data, pp. 261-272, 2000.

[13] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In Proc. the 2002 ACM SIGMOD international conference on Management of data, pp. 49-60, 2002.

[14] S. Chandrasekaran and M. J. Franklin. PSoup: a system for streaming queries over streaming data. The VLDB Journal, 12(2):140-156, 2003.

[15] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom. Adaptive Ordering of Pipelined Stream Filters. In Proc. of SIGMOD 2004, June 2004

[16] R. C. Fernandez, M. Migliavacca, E. Kalyvianaki, P. Pietzuch. Integrating scale out and fault tolerance in stream processing using operator state management. In Proc. ACM SIGMOD, pp. 725-736, 2013.

[17] Y. Wu and K. Tan. ChronoStream: Elastic stateful stream computation in the cloud. In Proc. ICDE, pp. 723-734, 2015.

[18] Resoure Description Framework.
http://www.w3.org/RDF/

[19] SPARQL Query Language for RDF.
http://www.w3.org/TR/rdf-sparql-query/

[20] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, M. Grossniklaus. Querying RDF streams with C-SPARQL. SIGMOD Record, Vol.39 No.1, pp.20–26, 2010.