Indian Journal of Science and Technology, *Vol 8(27), DOI:10.17485/ijst/2015/v8i27/74930, October, 2015*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Timestamp Embedding Query Stream Processing Engine

M. Ananthi[1]* and M. R. Sumalatha[2]

[1]Department of Information Technology, Sri Sairam Engineering College, Anna University, Chennai - 600 044, Tamil Nadu, India; ananthi.it@sairam.edu.in
[2]Department of Information Technology, Anna University, Chennai - 600 025, Tamil Nadu, India; sumalatha@annauniv.edu

Abstract

Background/Objectives: To improve the performance of the streaming data through RDF vocabulary and multi query optimization. **Methods/Statistical Analysis**: The scalability and performance of the query processor is improved with the use of native timestamp embedding query processor with the help of semantic approach. This offers vibrant, scalable services for continuous query processing. Continuous data is controlled through dynamic sliding window for incremental evaluation of streaming data. Static queries are processed continuously and all processing is done within main memory. Semantic sensor web is used to process sensor data semantically by query processing engine and metadata will be stored as triple stores. **Findings**: The continuous stream of queries is properly handled by the dynamic sliding window and sent to the query optimizer for optimization. The triple formatted outcomes are warehoused into OWL, which stores and adopt the relational database format. The data conversion will be done by the data transformer and finally results will be realized as stream of data. In this proposed model, the RDF vocabulary has been redesigned by adding timestamps along with incoming triples. The classes and properties are depicted in terms of the triples (Subject S, Object O and Predicate P) and are designed as OWL. Data sets are split into smaller parts for continuous updates and collect them into massive data sets so as to obtain faster insertion rates. It takes less time to process temporal, complex and multi queries in proposed system. Since timestamp is attached as one of the object in the triples, it is easy to retrieve and execute timestamp embedding queries. The proposed system is appropriate for all types of queries and proves efficiency in execution time. **Application/Improvement:** Temporal, complex and multi join queries are executed and evaluated for weather monitoring system.

Keywords: OWL, Semantic Web, Sliding Window, Stream Processor, Timestamp Data

1. Introduction

Sensor-web integrates large-scale sensor networks through the web. Data composed from various sensors and other sources is called linked data. There are many systems which have been established to process, accomplish and envision sensor data streams. But the integration of semantic web and dynamic streams of data still lack in various QoS parameters as adapting continuous query processing is difficult to achieve. Data centers are required to keep the latest versions of sampling data as well as historical data. Triple storages (RDF) maintain historical information. The challenges involved in the existing systems are: 1. Triple storages are unable to handle high insertion rates (time to fetch data from all input sources and time to insert the output into the triple storages) efficiently and are not able to store ancient data. 2. Existing query processing engines are not supporting efficient multiple query processing. 3. Scheduling and hosting the synchronous threads for handling additional tasks lead to overhead problem and are not scalable. Based on the perspective of the existing problems, the objective of the proposed work is designed. The goals of the proposed work are to design a timestamp embedding stream processor and to develop a dynamic sliding window

*Author for correspondence

for incremental evaluation of triple-based windowing operator to improve the scalability and performance.

2. State of the Art

Data heterogeneity and software heterogeneity are two types of heterogeneity. Data heterogeneity is retrieving data from diverse sensors. Software heterogeneity is having the same type of sensors managed by different proprietary software. PIPEFLOW process[1] with complex event processing is introduced for scalable management of stream processing. Various query processing engines with PIPEFLOW process is analysed. Various challenges and issues in LOD[2] (Linked Open Data) are discussed. LOD tools and applications are given. Accurate query answering is a problem in semantic linked data. Complex query processing is difficult in LOD. Query Processing and query optimization is one of the challenging issues in data stream management system[3]. RDF queries on cloud[4] were enhanced using Map-Reduce technique (Apache Hadoop) on semantic web triple models. RAPID+ technique[5] (extension of Apache Pig system) was developed for optimizing queries on RDF data models using an algebraic approach. Time-affixed OWL[6] and TSPARQL is proposed and designed to improve query processing of real time data. Time is attached in the ontology design and RDF vocabulary is enhanced. The complexity in the retrieval of historical data is reduced. A data acquisition framework[7] was designed to query sensor data from heterogeneous applications. Efficient heuristic solutions are suggested for various query types and query mixes using multiple query optimizations. But the Optimal result was not scalable for huge database instances and semantic web was not used as the processing was done only in the network. Linked Stream Middleware[8] was developed to process dynamic online sensor data and to easily access any type of resource on the web. Performance bottleneck in mapping of relational to triples and query rewriter [relational to SPARQL] by virtuoso was done. Stream Cloud[9] is a scalable and adaptable stream processing engine for processing huge volume of data. Elastic protocols are intended to exhibit low invasiveness and to regulate the incoming load. Heuristic algorithm was suggested[10] to improve the scalability of multi-query optimization for SPARQL. Multiple SPARQL queries were tried to solve NP-hard problems by discovering the common substructures and a cost model proposed to compare with the candidate execution plans. CQELS[11] an adaptive stream query processor engine developed to execute streaming data. Performance of continuous queries was analysed. It was found that feed rates are higher than insertion rates and hence are not designed for write intensive applications. It is also not scalable when the number of tasks increases. Sensor Stream[12] is a Semantic Real-time Stream Management Systems with an ontology-based architecture. This is used to efficiently execute the queries of streaming XML data. Maintaining and constantly updating the window overloads the system and hence system performance needs to be measured properly. Map-Reduce technique[13] was used to reduce the processing time of join operations and dynamic optimization was introduced by eliminating the intermediate results. The various techniques of Hadoop clusters were analysed. Semantics Management of Streaming Data[14] is an RDF- compatible model developed with semantic stream in which a triple storage optimized function was presented. Query optimizer was not established and queries were not estimated. Various researches have been conducted on streams of sensor data and other data sources[15]. Triple T indexing[16] model was proposed to observe the processing time of join operations in RDF. A model was introduced to predict the number of I/O required for the join conditions. The analysis was done for plain-text based data and not for time related data. Archiving and Querying streams of sensor data which support multisensory inputs and multithreading ARQ[17] were developed. Performance of serialization was analyzed by considering indexing and querying using RDBMS and SQL. Semantic data was not deliberated.

The summary of the survey shows that triple storages are unable to handle high insertion rates efficiently and are not able to store ancient data and existing query processing engines are not supporting efficient multiple query processing. From the issues, it is proposed to design an efficient query processor that handles triples and supports multi query processing.

3. Timestamp Embedding Stream Processor

Data stream management system produces data continuously which has high input rate. This continuously arriving data need to be processed immediately. Static queries are processed continuously and all processing is done with the main memory. It is a very challenging task to efficiently process the incoming streaming data as

it has to produce the query outcome instantly. Weather monitoring retrieved through sensors is considered in this work. A dynamic sliding window is proposed for incremental evaluation of triple-based windowing operator and a query processor is designed to efficiently progress streaming data. Timestamp is embedded with stream processor because streaming data are time variant. Semantic web is used to effectively process sensor data. Semantic sensor web is the process used to integrate sensor data and semantic web.

3.1 Dynamic Stream Processor

Data is collected from various heterogeneous sources, in the form of XML or tables. The data is then transformed into semantic form, using semantic converter in the form of OWL representation. Then, the semantic data will be further processed by the query processing engine and the metadata will be stored in databases.

Algorithm 1 Query Processing Engine QPE (Q,T)

Input: Queries Q, Time T
Output: Query result R
1: Incoming queries Q
2: Split queries
sq1,sq2,sq3,….,sqn ∈ Q
3: **begin**
4: t_s – start time
//Create dynamic sliding window
5: **if** $T \geq t_s$ then
6: $W(S,t_s,T)= w_l(S,t_s,T)$
7: **else**
8: $W(S,t_s,T)= 0$
9: **end**
9: Assign $W(S,t_s,T) \rightarrow Q_{opt}(S)$
10: **begin**
11: $EQ \leftarrow Q_{opt}(S)$
12: Execute EQ
13: Store $EQ \rightarrow Q_c$
14: **end**
15: Join Q_{c1}, $Q_{c2,}$ $Q_{c3,...}$
16: Remove duplicates
17: $R \leftarrow$ Aggregate Q_c
18: Return the result R

The queries are collected from various users continuously. The continuous stream of queries is properly handled by the dynamic sliding window and will be sent to the query optimizer, which optimizes the queries in a cost-efficient manner. Then the queries are executed by SPARQL processing engine. The triple format outcomes are warehoused into OWL stores and adapted into user-intuitive format or relational database format. The data conversion will be done by the data transformer. Finally, results will be realized as stream of data. They are Pull Based Queries and Push Based Queries. Pull Based Queries are fixed queries and mainly used to store and process ancient data. Push Based Queries are unremitting queries. Here, both Pull Based and Push Based queries are considered.

Proposed Query processing engine algorithm is shown in algorithm 1. Incoming queries are split into sub queries sq1, sq2… (Step 2). Starting time is represented as t_s, T is the data arrival time. If current time is greater than starting time, continuous stream of data is received. Create a dynamic sliding window to handle incoming streaming rate (Step 5-8). Optimize the query to process incoming streaming data using Q_{opt} (S). Execute optimized query EQ and store the cost of query in Qc (Step 11-13). Finally, join all the optimized sub queries by aggregating the executed queries and removing duplicates (Step 15-17).

3.2 Dynamic Sliding Window

Dynamic Sliding window is used to process a continuous stream of queries. In Dynamic Sliding window, some of the tuples expire. Sliding window operators in Continuous Query Language (CQL) are tuple-based, time-based and partition-based sliding windows.

Definition 1 (Tuple-based sliding window)

At time T, the window contains the last m tuples of S. ie., S[Rows m]

$$R(t) = S[m],\ t \leq T$$

Where m is the number of tuples and t is the largest timestamp.

Definition 2 (Time-based windows)

The stream of data S retrieved at the specified time T. ie. S [Range t]

$$S(t) = S[\text{Range } t],\ s \in t–T,\ t \in T$$

Definition 3 (Partitioned windows)

It is a collection of sub streams based on the range of attributes A1,A2,..Ak.ie. [Partition S By A1,….Ak Rows N]

$$S(t) = S\{A1, A2, \ldots. Ak\}$$

Sliding windows W (S) have fixed length and are bound by a predefined interval (t1 and t2). Only some of

the tuples expire at a specified time T. There is an overlap between sliding windows. It is symbolized as

$$W(S, t1, t2) = S(T), t1 \leq T \leq t2$$

Here, the window is bound by the time specified between t1 and t2. Another window is called the tumbling window, which moves at a fixed interval of length, as the window size (w) is predetermined. All tuples within the window expire at specified time. i.e.

$$W(S, t0, T) = W(S, w, t0, T)$$

The above two window models were mostly used in the existing systems. There is a window, called Landmark window. In the Landmark window Wl(S), the beginning time is fixed and bound with the window. The other end is movable. It is denoted as

$$W(S, tl, T) = \begin{cases} Wl(S, tl, T), T \geq tl \\ 0, \quad T < tl \end{cases}$$

Dynamic sliding window model is proposed to process continuous stream of queries. Since the window size is vibrant, the window model intended is based on the landmark and sliding window concepts. The Dynamic Sliding Window W(S) is,

ts – Start time (fixed)

$$W(S, ts, T) = wl(S, ts, T), T \geq ts$$

$$wl(S, t1, t2) = \begin{cases} S(T), t1 \leq T \leq t2 \\ 0, \quad T < ts \end{cases}$$

The ultimate aim of the proposed work is to split the data sets into smaller parts for continuous updates and collect them into massive data sets so as to obtain faster insertion rates. Then high-update rate data is filtered and aggregated before storing into triple storages to reduce number of pull-outs. Persistent storages are used for keeping dropout elements from the window.

3.3 Semantic Approach

OWL (Ontology Web Language) is used for semantic representation. It is represented in the forms of RDF (Resource Description Framework), i.e. triples (subject, predicate, object). RDF, time and stream are not compatible in the existing system. So, it is proposed to attach the time along with the RDF triples which is easy to recognize the data semantically. Time-affixed OWL[16] represented from the Proof of Concept has been taken as the basis. The model proposed[16] was not evaluated using SPARQL. Here, the date and time are attached with the RDF representation itself. In the existing middleware system, the timestamps are attached separately. In the proposed model, the RDF vocabulary has been redesigned by adding timestamps. The class and its properties are depicted in terms of the triples (Subject S, Object O and Predicate P) and are designed in OWL. OWL is bigger and better than RDFS. It works efficiently with the database queries especially temporal queries. OWL has a very sophisticated vocabulary, rigidity and better reusability. Thus, the use of OWL instead of RDF will enhance the working of the proposed model by reducing the time required for executing temporal queries. According to Kanwei analysis[16], B+ Tree indexing mechanism is used to represent data in the form of Triple format. The triple format designed for stream of sensor data is disjointed into three buckets, a subject bucket, a predicate bucket and an object bucket. Each data is timestamp related. The sample triple diagram is represented in the Figure 1.

The subject "Reading" can be represented as
(Reading, associated with, Location),
(Reading, contains, temperature),
(Reading, contains, humidity),
(Reading, has, time).

4. Query Implementation and Evaluation

Date as well as time is considered to be one of the values in the readings. A sample query was tested, evaluated and represented in the previous work[6]. Weather readings have been taken as an experiment. The time-affixed OWL model is developed and tested in Protege. The queries are

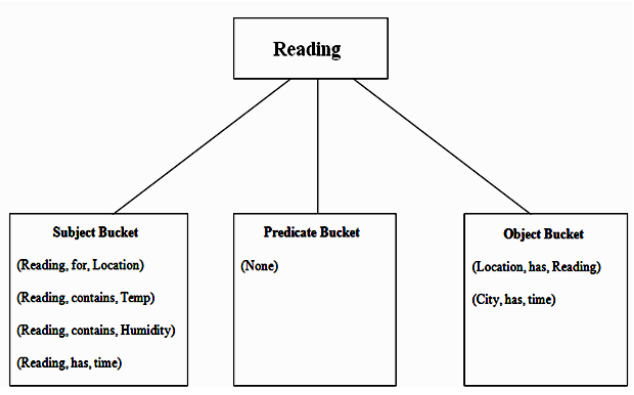

Figure 1. Triple diagram.

executed using the SPARQL query language. Extended SPARQL is proposed to process the time-affixed OWL. There are three types of queries in SPARQL. Point-based queries, range-based queries and timestamp-based queries are designed and executed. Static data is considered to be a sample to test the proposed model. The temporal, time-based data is represented in the triple pattern where query processing is done with TOWL. Some of the sample queries are designed and given below:

- Query 1: What is the temperature at Bangalore on 14-04-2015 at 5:00 pm?
  S= {[(?loc, :has, ?temp)] | ?time= "2015-04-14T17:00:00Z" && ?loc= "Bangalore" }
  Where{[?s, p, ?0] | ?time= timestamp && ?s= Location }
- Query 2: What are all the days in which Chennai had temperature less than 35 degree Celsius?
  S= {(Loc, :with, ?date) ∈ S | ?temp<35}
  Where S={(s, p, ?o) ∈ S | ? t < 35}
- Query 3: Which location has the highest temperature on 14/04/2015?
  S= { (?loc, :has, ?temp) ∈ S } { (?temp, on, date) ∈ S | MAX(?temp) }
  Where S = {(?s, p, ?o) ∈ S} { (s, p, ?o) ∈ S | MAX(?o)}
- Query 4: What is the Current humidity at Mumbai?
  S={(loc, :has, ?temp) ∈ S | ?time = systemDateTime}
  Where S = { (s, p, ?o) ∈ S | ?t=systime}
- Query 5: Which locations have temperature greater than Chennai?
  S= {(?loc, has, ?temp) Π S | Π [?temp > (loc, has, ?temp)] }
  Where S= {(?s, p, ?o) ∈ S | Π [?o> (s, p, ?o)]}

Assume a data set D. Let S (D), P (D) and O (D) be the set of subject, predicate and object atoms in D respectively. For joining two triples of (S, P, O), the different combinations are SO, SP, PO, PS, OP, OS. Consider the basic triple pattern (a, b,?c) ^ (?c, d, e) where a, b, d, e are distinct atoms. In this 'c' represents the temperature, 'a' represents the location and 'e' denotes the specified time. The query is "what is the temperature of a particular location at a specified time?" First the left pattern is evaluated and then the result is evaluated according to the condition in the right pattern. In this 'b' acts as an object in the left pattern but in the right pattern it acts as a subject. This is called, subject-object combination. This query involves the matching of the left pattern (object) with right pattern (subject) based on the join on c. When the date timestamp is made a part of the ontology, the time required for computing the required result is less since the parameter used for the query evaluation is defined as a subject in the ontology. Thus the overall computational performance of the Timestamp embedding OWL is efficient.

5. Results and Discussion

Timestamp embedding query processor for streaming data has been implemented using dynamic sliding window and dynamic stream processor with semantic approach. Time taken to process the queries is shown in the graph. Since time is made as a property in the ontology developed for our sample, the queries which involve date and time as parameters get evaluated and executed. This result is shown in the following graphs with the help of the sample queries. Query 1 is used to find the temperature in a particular location on a specified date and time. This is an example for multi-join query. First the date and time associated with the location is evaluated and then the result is joined with the temperature of the particular location. In order to process the query, date and time are used as parameters in the filter statement. Query 2 is used to list the dates on which the temperature of a particular location is less than the specified temperature. This is simple query which displays the dates by filtering based on the temperature. In case of TOWL, the date and time can be evaluated easily as it is a part of the ontology. Query 3 is to find a location which has the highest temperature on a specified date. This query performs an aggregate function of finding the average. First, all the temperatures recorded on that particular day is retrieved and then the average of the recorded temperature is displayed. This is also a temporal query as it involves date as its parameter for filtering. Thus it is an example of a complex query. TOWL efficiently processes this query. Query 4 is to find the current humidity in Mumbai. The Query result must be accurate and instant result is required. Time taken to execute this query is lesser than other queries and this shows that the proposed method is efficient. Query 5 is to find the locations which have temperatures higher than that in Chennai. This is a complex and multi-join query and hence takes little bit more time than other queries. Time variation is very minimal. As the number of triples increases the variation in time also increases which is graphically shown in the Figure 2. The proposed system is appropriate for all types of queries and proves efficiency in execution time.

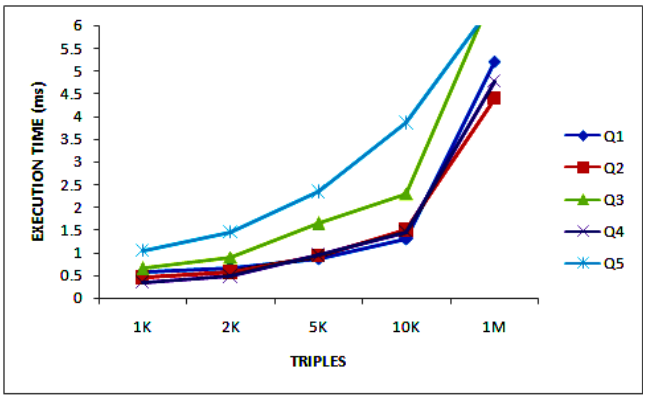

Figure 2. Query execution time vs. number of triples.

6. Conclusions and Future Work

Integration of semantic web and dynamic stream of data lacks in scalability, throughput and performance. An efficient query processing engine is needed to process the real-time data. The algorithm for query processing engine and dynamic sliding window is proposed. An efficient ontology is proposed to represent and query the time-series data. Initially, the time-affixed OWL and extended SPARQL are designed and evaluated. A time-affixed OWL is implemented to improve the RDF vocabulary using the semantic web approach and to develop a dynamic sliding window for incremental evaluation of triple-based windowing operators. The design is focused to reduce the complexity in retrieving historical data from classes and properties and an Effective Stream Processing Engine designed by properly scheduling and optimizing the stream of query processing in order to improve the scalability and performance. The execution time of the proposed TOWL is improved even when there is an increase, in number of triples which shows the proposed system outperforms in executing the query.

7. References

1. Omran S, Stefan H, Kai-Uwe S. Complex event processing on linked stream data. Springer-Verlag Berlin Heidelberg. 2015 Jul; 15(2):119–29.
2. Shah K, Fouzia J, Mashwani SR, Alam I. Linked open data: Towards the realization of semantic web - A review. Indian Journal of Science and Technology. 2014 Jun; 7(6):745–64.
3. Andreas H, et al. Linked data management. Emerging Directions in Database Systems and Applications. Chapman and Hall/CRC; 2014.
4. Kim H, Ravindra P, Anyanwu K. Optimizing RDF(S) queries on cloud platforms. International World Wide Web Conference Committee (WWW 2013); Companion Brazil. 2013. p. 261–4.
5. Kemafor A, Hyeongsik K, Padmashree R. Algebraic optimization for processing graph pattern queries in the cloud. IEEE Internet Computing. 2013 Mar-Apr; 17(2):52–61.
6. Ananthi M, Sumalatha MR. Efficient management of semantic streaming data using TOWL. 4th International Conference on Computer and Communication Technology (ICCCT); Allahabad. 2013 Sep 20-22. p. 247–51.
7. Riahi M, Papaioannou TG, Trummer I, Aberer K. Utility-driven data acquisition in participatory sensing. Proc of the 16th International Conference on Extending Database Technology. EDBT/ICDT; 2013. p. 251–62.
8. Le-Phuoc D, Nguyen-mau HQ, Parreira JX, Hauswirth M. Middleware framework for scalable management of linked stream. Journal of Web Semantics: Science, Services and Agents on the World Wide Web. 2012 Nov; 16(11):42–51.
9. Gulisano V, Jimenez-Peris R, Patiño M, Soriente C, Valduriez P. Streamcloud: An elastic and scalable data streaming system. IEEE Transactions on Parallel and Distributed Systems. 2012; 23(12):2351–65.
10. Le W, Kementsietsidis A, Duan S, Li F. Scalable multi-query optimization for SPARQL. 28th IEEE International Conference on Data Engineering; 2012 Apr. p. 1–12.
11. Le Phuoc D, Dao-Tran M, Parreira JX, Hauswirth M. A native and adaptive approach for unified processing of linked streams and linked data. ISWC. 2011; 7031:370–88.
12. Spanos DE, Stavrou P, Mitrou N. Sensor Stream - A semantic realtime stream management systems. Int J of Ad Hoc and Ubiquitous Computing. 2012; 11(2/3):178–93.
13. Ravindra P, Hong S, Kim HS, Anyanwu K. Efficient processing of RDF graph pattern matching on MapReduce Platforms. Proc of the second international workshop on Data intensive computing in the clouds; 2011. p. 13–20.
14. Rodriguez A, McGrath R, Liu Y, Myers J. Semantics management of streaming data. Proc Semantic Sensor Networks; 2009. p. 80–95.
15. Goebel V. Data stream management systems for sensor networks [Tutorial talk]. Norway: University of Oslo; 2010.
16. Li K. Cost analysis of joins in RDF query processing using the tripleT index. Emory University; 2009.
17. Pu KQ, Zhu Y. Fast archieving and querying of heterogenous sensor data stream. IEEE 2nd Int Conf on Digital Telecommunications; San Jose. 2007 Jul 1-5. p. 28.