Proactive Replication of Dynamic Linked Data for Scalable RDF Stream Processing

Sejin Chun, Jooik Jung, Xiongnan Jin, Seungjun Yoon, and Kyong-Ho Lee

Department of Computer science, Yonsei University, Seoul, Republic of Korea
{sjchun,jijung,wnkim,sjyoon}@icl.yonsei.ac.kr, khlee89@yonsei.ac.kr

Abstract. In this paper, we propose a scalable method of proactively replicating a subset of remote datasets for RDF Stream Processing. Our solution achieves a fast query processing by maintaining the replicated data up-to-date before query evaluation. To construct the replication process effectively, we present an update estimation model to handle the changes in updates over time. With the update estimation model, we re-construct the replication process in response to the outdated data. Finally, we conduct exhaustive tests with a real-world dataset to verify our solution.

1 Background

RDF Stream Processing (RSP)[1] produces possibly continuous answers to queries over RDF streams and background data, i.e., Linked Data. The current RSP engines such as C-SPARQL[1] and CQELS[2] support operations on RDF streams as well as background data by exploiting their RSP query languages. The RSP languages allow querying RDF streams as well as datasets in remote SPARQL endpoints using a `SERVICE` clause.

Due to the limitations of SPARQL endpoints such as availability and performance, RSP engines are required to optimize the evaluation of the query with `SERVICE` clauses. Since the evaluation time increases in proportion to the number of remote services, the query response may be delayed significantly. To make a fast response, many researches exploit a materialized view (MV), which caches a snapshot returned from a remote SPARQL endpoint. So, an RSP engine pulls the cached data from MV, so that the number of service invocations can be reduced at query evaluation.

In comparison with prior works in continuous query evaluation over streams and background data, several techniques utilize only static data or quasi-static data for local and centralized repositories. Recently, the authors of [3] propose an optimization method to limit the number of remote services at query evaluation, according to user-defined QoS constraints. However, their method generates one or more remote service invocations at specific query evaluation whenever the replicated data corresponding to them must be refreshed. Consequently, they may not guarantee a fast response time.

[1] https://www.w3.org/community/rsp/

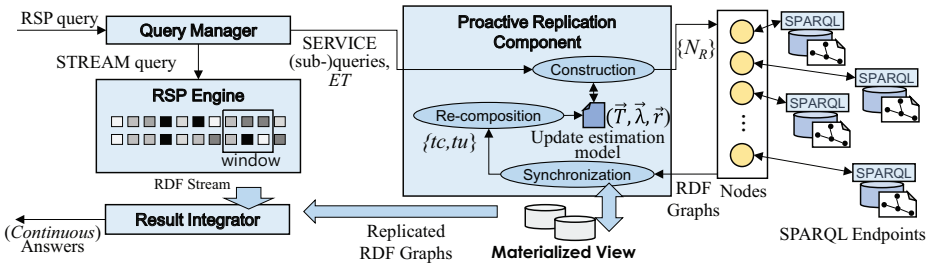

Fig. 1. The proposed system

2 Solution

Our solution presents a proactive replication of Linked Data for RSP. The proposed solution refreshes the replicated data retrieved from a SPARQL endpoint before query evaluation. In other words, we maintain the replicated data up-to-date before joining stream data with remote data. Thus, we achieve a fast query processing because we do not require any invocations to the endpoints at every query evaluation while maintaining a high accuracy.

Figure 1 illustrates the proposed system. Given an RSP query that joins RDF streams with `SERVICE` patterns, a query manager accepts the query as an input and divides it into two queries: `STREAM` and `SERVICE` queries. The `STREAM` query should be delegated to an RSP engine like C-SPARQL, and `SERVICE` (sub-)queries should be delivered to a proactive replication component (PR). An RSP engine registers the `STREAM` query and evaluates it continuously. Meanwhile, from the `SERVICE` (sub-)queries, PR constructs a replication process N_R, in which each instance invokes a remote service and materializes the result to MV. Lastly, a result integrator combines the results obtained from RSP and PR, and produces answers continuously.

Specifically, PR consists of three phases: construction, re-composition, and synchronization. In the construction phase, PR constitutes N_R with an update estimation model. Each instance of N_R is assigned to a node in order to obtain a subset of remote RDF data through a SPARQL endpoint. To model various changes in the number of updates λ over time, our update estimation model is based on the *inhomogeneous recurrent piecewise constant process* [4]. The underlying assumption of such process is that λ repeats every Q time unit, in other words, $\lambda(T) = \lambda(T + Q)$ for all time periods T. Thus, we construct an initial version of an update process N_U by assigning λ to a given time interval.

With the initial version of N_U, we create and deploy the instances of N_R based on a set of evaluation time ET to select stream data. Let a time-based sliding window $\mathbb{W}$ consist of (α, β), where α is a width of the window and β is a slide as the gap between the opening time instants of consecutive windows. Given a query q that contains one or more $\mathbb{W}$s, we compute $ET = \{\tau_1, \cdots, \tau_n\}$ for q, where each τ indicates the evaluation time for each window W_n of $\mathbb{W}$. Therefore, we determine the number of instances of N_R and their positions by N_U and ET.

Given a time interval Q, the solution mappings μ of a `SERVICE` pattern and the update estimation model $= (T, \lambda)$ for all time periods T, we define a repli-

cation process N_R of μ in the following:

$$\boldsymbol{N_R(\mu) = (T, \lambda, r(\mu))} \tag{1}$$

Where a vector represents an effective replication instance $r(\mu)$ with the λ value for each time interval T, and each $r(\mu)$ is composed of half-opened intervals of the form $[s, f)$. The start time s is the time at which the `SERVICE` patterns corresponding to μ executes and the finish time f is the time of replicating the solution mappings μ retrieved from the endpoint.

In the synchronization phase, PR receives the set of solution mappings retrieved by the instance of N_R and replicates them into MV. To renew the update estimation model, the information about the replicated data (i.e., whether the data changes(t_c) or not(t_u) is transferred and computed for new λ over a time period.

In the recomposition phase, PR re-constructs N_R using a new λ and a cost metric at time t such as $M(t)$ and $G(t)$. In detail, $M(t)$ is defined as the number of updates being missed from MV at time t. Larger $M(t)$ deteriorates the freshness of MV, and decreases the accuracy of the answer. $G(t)$ is defined as the number of replication instances in which the result of `SERVICE` patterns is equivalent to the duplicated data in the prior release in [0,t]. Thus, reducing $G(t)$ improves the performance of maintaining MV in terms of stability.

To derive new λ from irregular invocations to endpoints, we use a maximum-likelihood estimator (MLE) [5]. The MLE computes the expected λ that has the highest probability of producing the observed set of changes, which are detected from accesses. Since each access to an endpoint can determine whether the requested dataset has been updated(t_c) or not(t_u), we estimate new λ without complete history of updates.

3 Evaluation

Experimental Setup. We developed our solution based on C-SPARQL. To compare with the state-of-the-art work, we implemented a process of maintaining MV [3]. In addition, we selected CQELS as a baseline method which performs generally better than C-SPARQL. We utilized a query Q6 and its related datasets from CityBench[2]. In addition, we extended the query by adding remote services that provide real-world parking information[3,4]. Here, to maintain the average response time of a service, e.g., $\leq$ 1s, consistently, we used subqueries, e.g., ⟨entityURI⟩ ?p ?o. Both the average of result sizes with 850kb and the number of results with 5000 records are approximately similar at every query evaluation.

Experimental Result. Figure 2 shows the average execution time of processing Q6 with varying the number of `SERVICE` patterns. On average, our method took five seconds less than the method of [3]. Specifically, the amount of reduced execution time .5 seconds for two services, 1 second for 4 services, 6s for 8 services, and 11s for 16 services, respectively. This improvement is due to that our

[2]https://github.com/CityBench/Benchmark

[3]https://www.parkwhiz.com

[4]http://lod.seoul.go.kr/

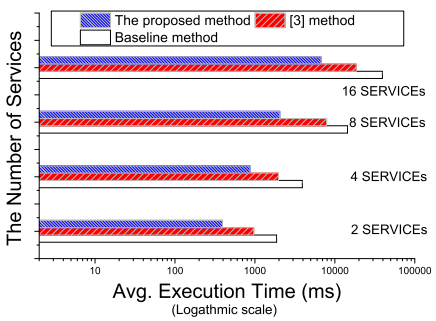

Fig. 2. The avg. execution time according to the number of SERVICE patterns

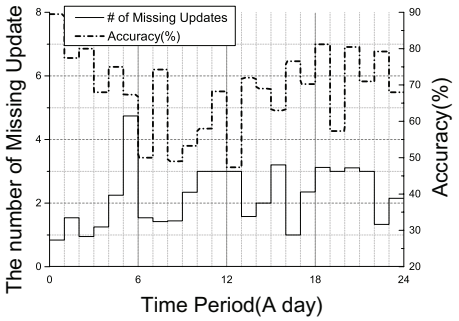

Fig. 3. Correlation between the accuracy and the number of missing updates

solution pulls the replicated data from MV at every query evaluation.

Figure 3 shows a correlation between the accuracy and the number of missing updates. Using parking information during a week, we checked how many updates were missing from MV. We then measured the accuracy of the replicated data using Jaccard Similarity, that is defined as the size of the intersection of the replicated and the answer sets divided by the size of the union of them. At each hour, the result has a higher accuracy and small number of missing updates, i.e., 00:00 to 05:00 and 06:00 to 24:00, whereas some cases have larger number of missing updates but the accuracy is also high, i.e., 05:00 to 06:00. At each hour, it has a higher accuracy and small number of missing updates. In addition, we utilized that the Pearson correlation coefficient method estimates the correlation which is a strength of relationship between the accuracy and the number of missing updates. The obtained value of the coefficient was -.234, which indicates that the correlation is weak. From this experiment, we learned that our solution of maintaining the replicated data up-to-date before query evaluation may not have a strong influence on the accuracy of the answer.

Acknowledgement. This work was supported by the ICT R&D program of MSIP/IITP, Republic of Korea. [B0101-16-1276, Access Network Control Techniques for Various IoT Services]

References

1. Barbieri, D. F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: C-SPARQL: SPARQL for continuous querying. In: WWW, pp. 1061-1062. ACM. (2009)
2. Le-Phuoc, D., Dao-Tran, M., Parreira, J. X., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: ISWC 2011, pp. 370-388 (2011).
3. Dehghanzadeh, S., DellAglio, D., Gao, S., Della Valle, E., Mileo, A., Bernstein, A.: Approximate continuous query answering over streams and dynamic linked data sets. In: ICWE 2015, pp. 307-325 (2015)
4. Bright, L., Gal, A., Raschid, L.: Adaptive pull-based policies for wide area data delivery. In: ACM Trans. Database Syst., Vol. 31, No. 2, pp. 631-671 (2006)
5. Cho, J., Garcia-Molina, H.: Estimating frequency of change. In: ACM Trans. on Internet Technology (TOIT), Vo. 3, No. 3, pp. 256-290 (2003).
6. Chun, S., Seo, S., Ro, W., Lee, K.-H.: Proactive Plan-Based Continuous Query Processing over Diverse SPARQL Endpoints," In: WI 2015, pp.161-164, 2015.