Adaptive Cache Replacement in Efficiently Querying Semantic Big Data

Usman Akhtar
Department of Computer Science & Engineering, Kyung Hee University
Yongin-si, Gyeonggi-do, South Korea
usman@oslab.khu.ac.kr

Sungyoung Lee
Department of Computer Science & Engineering, Kyung Hee University
Yongin-si, Gyeonggi-do, South Korea
sylee@oslab.khu.ac.kr

***Abstract*—This paper addresses the problem of querying Knowledge bases (KBs) that store semantic big data. For efficiently querying data the most important factor is the cache replacement policy, which determines the overall query response. As cache is limited in size, less frequently accessed data should be removed to provide more space to hot triples (frequently accessed). Moreover, performance bottleneck of *triplestore*, makes real-world application difficult. To achieve a closer performance similar to RDBMS, we have proposed an *Adaptive Cache Replacement (ACR)* policy that predict the hot triples from the query log. Our proposed algorithm effectively replaces cache with high accuracy. To implement the cache replacement policy, we have applied *exponential smoothing*, a forecast method, to collect most frequently accessed triples. The evaluation result shows that the proposed scheme outperforms the existing cache replacement policies, such as *LRU (least recently used)* and *LFU (least frequently used)*, in terms of higher hit rates and less time overhead.**

***Keywords*-RDF Caching; Linked Data; Exponential Smoothing; Cache Replacement;**

I. Introduction

The heap of structured data published over the Internet is increasing i.e., Linked Data [1]. Linked Data is a global information space for representing and connecting data structurally. The format of Linked Data is encoded as RDF[1] which consists of subject, predicate, and an object and is stored in the Triplestore[2]. RDF is widely used as an information model for vast semantic data. However, in RDF data model the querying complexity is higher than the relational data model. As the SPARQL[3] is standard language to query RDF dataset. To access the data, SPARQL service is deployed on each knowledge base which use the HTTP bindings. The main part of the SPARQL language is Web Services Description Language (WSDL)[4] that describe the means for conveying queries and results to the processing service. Currently, widely used RDF datasets such as DBpedia[5] produces abundant request from diverse applications [2]. Nowadays, the amount of the semantic data is growing rapidly, therefore for efficient query processing and caching [2] is required. So caching is used to leverage the query processing on the Triplestore and the data is present in its cache the request is sent immediately (also called *cache hits*) [3]. Many caching techniques have been developed such as LRU [4] and LFU [5] for relational databases. The underlying structure of the big semantic data is different from the relational databases. In recent years, a lot of non-relational Triplestore [6] are emerging. The caching algorithm design for relational databases is not applicable to Triplestore [7]. In the RDF triplestore, some of the records are "hot" (frequently accessed by the application) and others are "cold" or seldom accessed. The performance depends on the number of factors such as hot records in the cache or residing in the memory for fast access [8]. Our work is motivated by the need for efficient query processing in the Triplestore. The following considerations drive our research:

(1) Access Workload: The performance of the Triplestore is a major challenge in real world practical application. The workload exhibits considerable access skew, for example, the product description in online store exhibits natural skew as most of the items are popular and frequently accessed than others [9]

(2) Overhead in caching: The major problem of cache is high overhead due to the proactive fetching [10], [11]. For example, cache policy such as LRU encounters 25% overhead on every record access [7].

In this paper, we introduce an approach to identify to identify the hot triples and develop adaptive cache replacement policy, which check the access frequency of highly retrieved triples. We first extract the query from the accessed log and use the exponential smoothing method to estimate the most frequently accessed triples. Our cache replacement evaluates the frequency of the cached queries and ignore the less frequent access triples. For the cache replacement, we choose exponential smoothing due to its higher accuracy and fewer error rates as shown in Figure 1. The standard error of smoothing is significantly less than the LRU approach and it is precise. Poor accuracy of the items is very crucial as miss-classified records reduce the in-memory hit rates.

The main contributions of this paper is summarized as follows:

[1] https://www.w3.org/RDF/
[2] https://jena.apache.org/
[3] https://www.w3.org/TR/rdf-sparql-query/
[4] https://www.w3.org/TR/rdf-sparql-protocol/
[5] https://www.dbpedia.org/

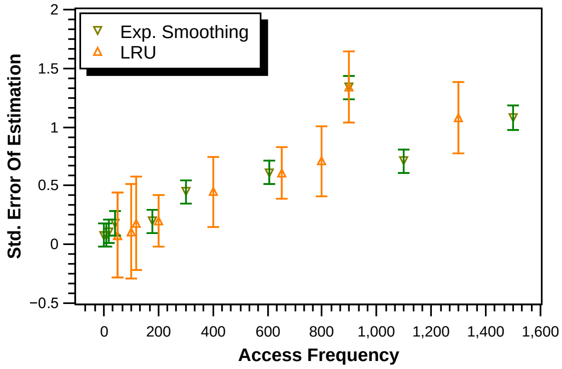

Figure 1: Showing the standard error rates of LRU and exponential smoothing.

1. We have developed an Adaptive Cache Replacement (ACR) algorithm to replace the triples with frequently accessed ones for higher hit rates.

2. We have utilized exponential smoothing, a forecast method to identify the number of frequently accessed triples. The results show that our proposed technique outperforms in terms of higher hit rates and less overhead.

The rest of the paper is organized as follows. We introduced our proposed methodology in section II, which briefly describes the identification of hot triples and cache replacement policy. In section III, we have performed the experiments on the real datasets and the results outperformed other state-of-the-art approaches. We concluded this paper in Section IV and discussed future research directions.

II. The Proposed Methodology

A. Overview

Our proposed approach illustrated in Figure 2 consists of the two main phases; offline and cache replacement phase. In the offline phase, we analyze and extract the frequently accessed triples. The query record will keep track of accessed queries and send for offline analysis to calculate the frequencies. If the query is not present in cache the result is transmitted from SPARQL endpoint and stored in the cache module. Cache module will maintain the frequently accessed triples, when the cache becomes full it will replace the triples.

B. Logging And Offline Analysis

Previously executed queries provide the valuable information and reflect the user's interest in data. Therefore, query log analysis is a method to extract relevant data from semantic big data cloud. We performed offline analysis to extract the previously issued queries. We have utilized publicly available query log provided by USEWOD2011 [12] challenge. The log[6] contain several months of usage

[6]https://httpd.apache.org/docs/1.3/logs.html

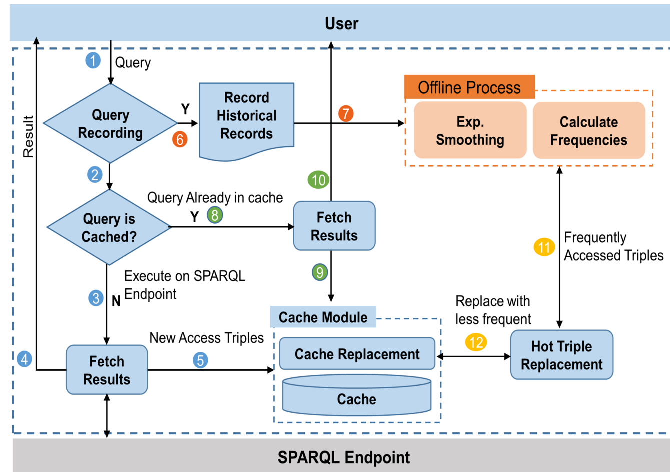

Figure 2: Proposed model for identifying hot triple for fast query processing

data such as DBpedia[7]. The format of the log consists of IP address and timestamps. The main goal is to extract the query from the log that was previously requested, include the time when a query was performed. We observed that the log contain duplicate and invalid queries. Linked Data consist of the set of triples such as $< S, P, O >$ where S represent the subject, P represent the predicate and O represent the object. Similarly, SPARQl query can be represented as:

$$Q = (query - type, patternP, solution - modifier) \quad (1)$$

In equation1, main part of the query is *P*, which contain patterns that match with the Linked Data. The *solution-modifier* performs aggregation, grouping and eliminating the duplicates. For the output of SPARQL query Q *query-type* determines the option which could be *SELECT*, *ASK*, *CONSTRUCT* and *DESCRIBE*. In the next section, we will discuss the use of the access log in determining the hot triples as well as the cache replacement strategy.

C. Identifying and Caching Hot Triples

The identification of hot triples using exponential smoothing, a forecast method to estimate the triples access frequencies. The frequencies are arranged in descending order, which also includes the hot triples stored in cache. To update the record, we proposed cache replacement strategy.

*1) **Exponential Smoothing** :* We have applied exponential smoothing a forecast method, to identify the hot triples and estimate their frequencies. The general formula of smoothing is as follows:

In equation2, E_t stands for the observation time t and x_t represents the observation in discrete time, α is a constant with the value (0-1). The high value of α gives significance to new observations. The reason behind choosing

[7]http://wiki.dbpedia.org/

Algorithm 1: Adaptive Cache Replacement (ACR)

Data: $(AccessLogL, HotDataSizeK)$;
Input : $Records$,$CachedTriples$
Output: $UpdatedcacheTriples$
$t_latest \leftarrow max(lastAccTime, cachedTripples)$;
$t_earliest \leftarrow min(lastAccTime, cachedTripples)$;
$estimation \leftarrow max(estimation, cachedTripples)$;
Function:
$ForwardAlgo(AccessLog, HotDataSize)$;
if $newAccessTriples\ in\ cache$ **then**
  $max(estimation, cachedTripples)$;
  $Calculate(AccessFrequencies, lastAccRecord)$;
  $Update(estimation, lastAccRecord)$;
  $Remove(lastAccTime < t\_earliest)$;
else
  $newAccessTriples$ not in $cache$;
  $Calculate(estimation, lastAccRecord)$;
  $Remove(lastAccTime$ is minimum in $cache)$;
  $addToCache(newAccessTriples)$;
end
return $UpdatedcacheTriples$;

exponential smoothing is its simplicity and high accuracy. The accuracy of estimation is often measured in terms of the standard deviation.

$$E_t = \alpha * x_t + (1 - \alpha) * E_{t-1} \tag{2}$$

Logging every record is not the optimal solution to estimate the access frequency as it degrade the performance of the system. We proposed our algorithm that classify the records as hot and cold using the exponential frequency. We scanned the logs from beginning to end point t_b. Upon encounter, the ACR algorithm update counter by using Equation 1. The scanning of the logs is still computationally expensive. We have applied the naive sampling approach, which does not store all the record but only the certain ones.

*2) **Cache Replacement**:* The size of the cache is limited, so it is essential for the Linked Data application to prioritize only the important data from the cache. When a user requests a data which is present in cache the task is accomplished. In this paper, we proposed an adaptive cache replacement scheme that only store the hot triples.

In our approach, we maintain the partial records for specific time period. Suppose, last observed time of the triple is denoted as $last\_time$, we only keep the estimation of $last\_time$. Our algorithm will show the cache hits if new access triples are in the cache. If the new access records are not in the cache (*cache miss*) then the proposed adaptive cache replacement update its estimation for the new triples. This approach will place the hot triples in the cache and replace with fewer access triples. In algorithm 1 we described the adaptive cache replacement for estimating the number of accessed triples using the exponential smoothing.

III. Evaluation

To evaluate the effectiveness of proposed approach, few experiments were performed on real datasets. The results outperform the current state-of-the-art cache replacement approaches.

Datasets: In our evaluation, real world queries we utilized provided by USEWOD 2014 challenge[8]. First the query logs were analyzed from SPARQL endpoints. The query log contains IP address, timestamp, query and userID. The valid queries were extracted (approx 198,000) from the log the syntax of query was checked according to SPARQL1.1 specification. The data was stored inside the Virtuoso server.

Performance Metrics: In this paper, well-know performance metrics such as hit rate was applied to compare our approach with LRU and LFU. The cache hit rate is computed as follows:

$$HitRate(HR) = \frac{\sum_{i=1}^{N} q_i}{N} \tag{3}$$

The hit rate is widely used as a standard for performance evaluation. The parameter q_i is a Boolean number which is used to calculate the hit rate. Whereas the N is a total number of the hit counts.

The comparison of Hit Rates: The proposed approach was compared with the traditional cache replacement algorithms such as LRU and LFU as shown in Figure 3. When the cache size increases, the performance of existing approach decreases which is not experienced in our proposed ACR. The performance of LFU was worst among other replacement scheme with the lowest hit rate. As the exponential smoothing has only one parameter α, the choice of α mostly depends on the performance of the cache. We have set the value to 0.05 due to the high accuracy of results obtained.

Time Overhead: Figure 4 shows the time of average hit rate of our proposed ACR with the state-of-the-art solutions. Most of the existing solution take almost 20 times delay as compared to the ACR.The proposed approach used the exponential smoothing which produced better results in terms of time overhead as compared to existing approaches. The cache is limited and more memory space is required for ACR algorithm, so there is a need for indexing algorithm to improve caches from our observation, we evaluated that the the server overhead is another area which needs to be analyzed further. As the pre-fetching of hot triples from SPARQL endpoint initiate high rate of overhead on server side. Since, the aim of this paper is to accelerate the query and propose an adaptive solution, so these overheads are beyond of the scope of the current work.

[8] http://usewod.org/usewod2014.html

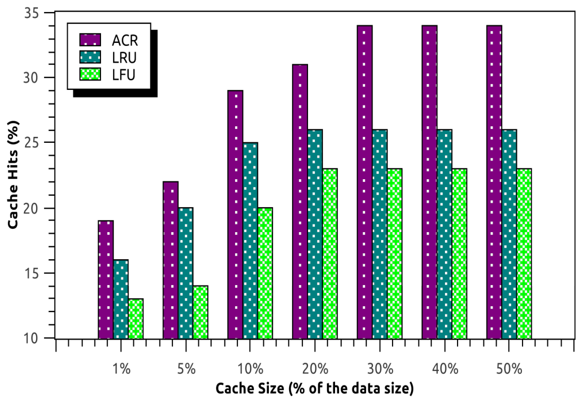

Figure 3: Hit Rate comparison with LRU and LFU

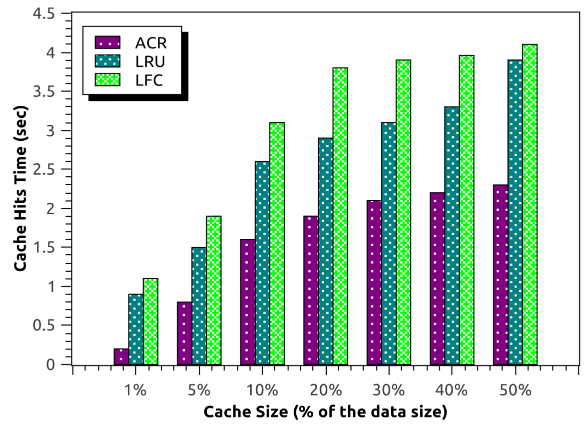

Figure 4: Time overhead comparison

IV. Conclusion

In this paper, we proposed an adaptive cache replacement policy to improve overall querying performance on big semantic data. Our proposed approach utilizes the exponential smoothing, a forecast method, to estimate the hot triples (i.e., frequently accessed). The process starts with extracting the queries from log. After the extraction phase, we applied forecast method to keep the frequent access triples in the cache. Our estimation based on the exponential smoothing was able to predict better result than existing LRU and LFU. The experimental results revealed superior performance of adaptive cache replacement approach as compared to existing approaches. In future, we will apply our approach in RDF archive to accelerate the processing on large knowledge bases.

V. Acknowledgments

This work was supported by Korea Research Fellowship program funded by the Ministry of Science, ICT and Future Planning through the National Research Foundation of Korea(NRF-2016H1D3A1938039) and this research was also supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2017-01629) supervised by the IITP(Institute for Information & communications Technology Promotion). This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2017-0-00655).

References

[1] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data-the story so far," *International journal on semantic web and information systems*, vol. 5, no. 3, pp. 1–22, 2009.

[2] M. Martin, J. Unbehauen, and S. Auer, "Improving the performance of semantic web applications with sparql query caching," in *Extended Semantic Web Conference*. Springer, 2010, pp. 304–318.

[3] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies," *ACM Computing Surveys (CSUR)*, vol. 35, no. 4, pp. 374–398, 2003.

[4] P. J. Denning, "The working set model for program behavior," *Communications of the ACM*, vol. 11, no. 5, pp. 323–333, 1968.

[5] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, "Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies," *IEEE transactions on Computers*, vol. 50, no. 12, pp. 1352–1361, 2001.

[6] K. Zeng, J. Yang, H. Wang, B. Shao, and Z. Wang, "A distributed graph engine for web scale rdf data," in *Proceedings of the VLDB Endowment*, vol. 6, no. 4. VLDB Endowment, 2013, pp. 265–276.

[7] J. J. Levandoski, P.-Å. Larson, and R. Stoica, "Identifying hot and cold data in main-memory databases," in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE, 2013, pp. 26–37.

[8] J. Lorey and F. Naumann, "Caching and prefetching strategies for sparql queries," in *Extended Semantic Web Conference*. Springer, 2013, pp. 46–65.

[9] N. Papailiou, D. Tsoumakos, P. Karras, and N. Koziris, "Graph-aware, workload-adaptive sparql query caching," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1777–1792.

[10] N. Megiddo and D. S. Modha, "Arc: A self-tuning, low overhead replacement cache." in *FAST*, vol. 3, no. 2003, 2003, pp. 115–130.

[11] E. J. O'neil, P. E. O'neil, and G. Weikum, "The lru-k page replacement algorithm for database disk buffering," *ACM SIGMOD Record*, vol. 22, no. 2, pp. 297–306, 1993.

[12] B. Berendt, L. Hollink, V. Hollink, M. Luczak-Rösch, K. Möller, and D. Vallet, "Usewod2011: 1st international workshop on usage analysis and the web of data," in *Proceedings of the 20th international conference companion on World wide web*. ACM, 2011, pp. 305–306.