Regular paper

Journal of the KIECS. pp. 109-116, vol. 16, no. 1, Feb. 28. 2021, t. 105, pISSN 1975-8170 | eISSN 2288-2189
http://dx.doi.org/10.13067/JKIECS.2021.16.1.109

# 융합 인덱싱 방법에 의한 조인 쿼리 성능 최적화

짜오티엔이[*] · 이용주[**]

# Join Query Performance Optimization Based on Convergence Indexing Method

Tianyi Zhao[*] · Yong-Ju Lee[**]

요 약

RDF(Resource Description Framework) 데이터 구조는 그래프로 모델링하기 때문에, 관계형 데이터베이스와 XML 기술의 기존 솔루션은 RDF 모델에 바로 적용하기 어렵다. 우리는 링크 데이터를 더욱 효과적으로 저장하고, 인덱스하고, 검색하기 위해 융합 인덱싱 방법을 제안한다. 이 방법은 HDD(Hard Disk Drive) 와 SSD(Solid State Drive) 디바이스에 기반한 하이브리드 스토리지 시스템을 사용하고, 불필요한 데이터를 필터하고 중간 결과를 정제하기 위해 분리된 필터 및 정제 인덱스 구조를 사용한다. 우리는 3개의 표준 조인 검색 알고리즘에 대한 성능 비교를 수행했는데, 실험 결과 제안된 방법이 Quad와 Darq와 같은 다른 기존 방법들에 비해 뛰어난 성능을 보인다.

ABSTRACT

Since RDF (Resource Description Framework) triples are modeled as graph, we cannot directly adopt existing solutions in relational databases and XML technology. In order to store, index, and query Linked Data more efficiently, we propose a convergence indexing method combined R*-tree and K-dimensional trees. This method uses a hybrid storage system based on HDD (Hard Disk Drive) and SSD (Solid State Drive) devices, and a separated filter and refinement index structure to filter unnecessary data and further refine the immediate result. We perform performance comparisons based on three standard join retrieval algorithms. The experimental results demonstrate that our method has achieved remarkable performance compared to other existing methods such as Quad and Darq.

키워드

Filter and Refinement Index Structure, Hybrid Storage System, Hash Join Algorithm, Linked Data, RDF.
필터 및 정제 인덱스 구조, 하이브리드 스토리지 시스템, 해시 조인 알고리즘, 링크드 데이터, RDF.

## Ⅰ. Introduction

Linked Data is a new form of distributed data on the Web which is especially suitable to be manipulated by machines and to share knowledge[1]. Linked Data uses RDF (Resource Description Framework) to create typed statements that link to anything in the world[2]. RDF is a description method of the graph database and uses the triple statement as the basic structure to describe the relationship between resources. A triple consists of three parts: subject, predicate, and

* 경북대학교 IT대학 컴퓨터학부(tianyi@knu.ac.kr)
** 교신저자 : 경북대학교 IT대학 컴퓨터학부
• 접 수 일 : 2020. 11. 11
• 수정완료일 : 2020. 12. 30
• 게재확정일 : 2021. 02. 17
• Received : Nov. 11. 2020, Revised : Dec. 30, 2020, Accepted : Feb. 17, 2021
• Corresponding Author : Yong-Ju Lee
School of Computer Science and Engineering, Kyungook National University,
Email : yongju@knu.ac.kr

object. Since RDF data structure is modeled as a graph, existing solutions such as relational databases and XML technologies are not suitable for RDF model[3]. Hence, more studies to store, index, and query Linked Data efficiently are needed.

The existing methods are mainly divided into two types. First, we can use efficient query processing convenience to maintain a separate copy of data in a centralized registry. We refer to this as the "*centralized method.*" Second, we can use link traversal to access distributed data dynamically. We refer to the "*distributed method.*" The centralized method collects data from known sources, merges the collected information, performs further processing, and finally stores the processed results in a centralized registry[4]. The advantage of this method provides the excellent query response time. But there are several disadvantages. Storing all data may be expensive. Users can only use Web data that has been copied to the registry. The distributed method performs queries over the multiple SPARQL (Simple Protocol and RDF Query Language)[5] endpoints that publishers provide for their Linked Datasets[6]. However, this method cannot guarantee that all publishers provide reliable SPARQL endpoints for their Linked Data.

We propose a *convergence method* between the centralized method and the distributed method. Our method consists of separated filter and refinement index structures with the hybrid storage system. Especially, our method aims to support efficient join query processing by quickly filtering valueless data.

The remainder of this paper is structured as follows. Section Ⅱ describes related work. Section Ⅲ proposes our convergence indexing method. Section Ⅳ describes the performance evaluation. Section Ⅴ summarizes and concludes our paper.

## Ⅱ. Related Works

A generally recognized method is the "exhaustive indexing" strategy to improve the query efficiency (e.g., Hexastore[7] and RDF-3x[8]). They all enumerate the various forms of triples that can be formed under multiple permutations between the subject, attribute, and object, and then build indexes for them. The index created in this way happens to be a six-fold index. That is, a B+-tree is established according to each permutation and combination. This strategy is derived from the Quad index method[9]. Shortcomings of this method: (1) Subject, attribute, attribute values of different triples may be repeated, and such repetitions will waste storage space. (2) Complex queries require a large number of table join operations. (3) When SPARQL queries are involved, the query cost of their join operations cannot be ignored.

With the development of Linked Data, more and more publishers are willing to express information with RDF data format. Many of these publishers provide SPARQL query interfaces and expressing data in the RDF data format to share data. These systems can independently receive SPARQL queries and calculate matches by customization interfaces. These independent RDF data sources are integrated into a system platform to form a distributed RDF data management system. Darq[10] is the first to discuss how to perform the SPARQL query processing on a distributed RDF data management system. This kind of design needs to decompose SPARQL queries into several sub-queries in advance and send them to their corresponding interfaces so that these corresponding RDF data sources can process the sub-queries and obtain partial results. After that, the system collects these partial results and obtains the final results by join operations. Therefore, this method produces a large number of intermediate results and consumes more time.

Bentley proposed the K-dimensional tree index structure. This data structure is widely used in the spatial index. The advantage of this tree is that it can solve logarithmic insertion, search, and deletion problems. It provides many query application protocols including scope, partial matching, nearest neighbor, intersection query, and so on[11]. Beckmann improved R-tree and proposed $R^*$-tree[12]. $R^*$-tree and R-tree[13] are identical in structure, and they are the same in tree construction, insertion, deletion, and retrieval. The difference lies in the following three points: the choice of insertion path, the split of nodes, and the forced re-insertion algorithm.

David analyzed and compared the hash join algorithm and the sort-merge join algorithm. The results clearly show that the hash join algorithm is superior to the sort-merge algorithm[14]. The nested-loop join algorithm[15] is one of the most commonly used join algorithms in traditional relational databases. It is logical simple, and the final result set is returned to the client in a pipeline way without waiting for results computed.

Due to the explosive growth of data volume, the large-capacity storage at low cost is a significant requirement for storage systems[16]. The storage device fundamentally determines a storage system's performance it relies on. HDD (Hard Disk Drive)[17] is still the main storage device used in the storage system. However, due to the limitation of the mechanical movement of the magnetic head, the random access performance of the disk is the bottleneck for a long time. SSD (Solid State Drive)[18] has emerged in recent years with the advantages of good random access performance, small size, and low energy consumption. Although there are still restrictions such as high price, small capacity, erase-before-write, and durability, it can form an excellent complement to HDD.

## III. Convergence Indexing Method

### 3.1 Hybrid Storage System

The hybrid storage system is to combine SSD with HDD. Frequently accessed data (so called hot data) can be stored on SSD and searched more rapidly than they are stored on HDD. This structure can improve the performance ratio, service life, reliability, capacity, and other indicators of the entire system. Hence, we adapt this hybrid storage system.

### 3.2 Separated Filter and Refinement Index Structure

In our convergence method, we first convert RDF data into hash values. In our structure, RDF tuples consist of points in $n$-dimensional data space represented by compressed hash values. We convert long string literals to hash values, which can significantly reduce memory stress. Then, we will store the data in the separated filter and refinement index structure. The purpose of this structure is to filter unnecessary data and further refine the immediate result to improve the join query performance. After filter and refinement processing, we will execute the hash join algorithm to obtain final results.

(1) Filter Phase

The first phase of our separated filter and refinement index structure is the filtering process. We use an $R^*$-tree[12] in this phase as shown in figure 1.

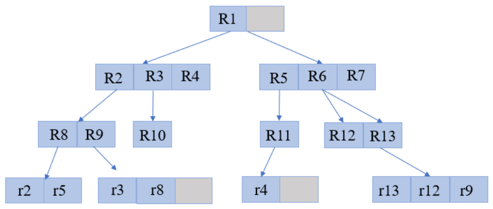


Fig. 1 $R^*$-tree

$R^*$-tree, the most popular variant of R-tree, is well suited for disk use, and it consists of leaf and non-leaf nodes. We use $R^*$-tree to reduce unnecessary spatial search as much as possible. With $R^*$-tree, we can quickly select the minimum bounding boxes (MBBs) that contain all possible RDF tuples that match the join triple query pattern.

All $R^*$-tree nodes are stored in HDD. Figure 2 shows the query performance for different SPARQL query types when $R^*$-tree is kept in SSD and HDD, respectively. We use three different join algorithms: hash join algorithm, nested-loop join algorithm, and sort-merge algorithm. We record the query time of different SPARQL query types when we use different join algorithms and calculate the average of each SPARQL query type to analyze the performance of $R^*$-tree on SSD and HDD. Experimental result shows that the performance difference of $R^*$-tree on SSD and HDD is not obvious. Thus, we use HDD for $R^*$-tree since the cost of SSD is higher than that of HDD.

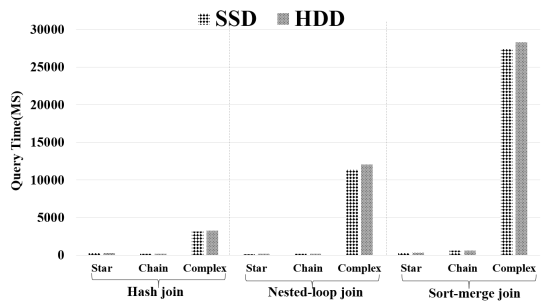


Fig. 2 $R^*$-tree performance on SSD and HDD

(2) Refinement Phase

The second phase is K-dimensional tree [10] groups, which play as refinement processing. Figure 3 shows K-dimensional tree. After identifying all candidate MBBs in the first phase, we perform the second phase, where further refine the results obtained in the previous filtering processing using K-dimensional trees. Then, we execute the hash join algorithm to calculate the data in K-dimensional trees and get the final results. We put K-dimensional trees in SSD since K-dimensional trees are great for memory because of their good storage utilization, fast search, and fast update. The random read performance for the hash join algorithm is also excellent in SSD.

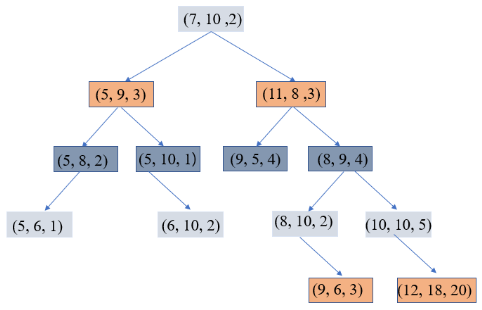


Fig. 3 K-dimensional tree

(3) Hash Join Algorithm

To determine which of three join algorithms (i.e., hash join, nested-loop join, and sort-merge join algorithms) has the best performance, we conduct experiments on three SPARQL query types: star type, chain type, and complex type. We record the query time of different SPARQL query types when using different join algorithms to analyze the performance of each join algorithm.

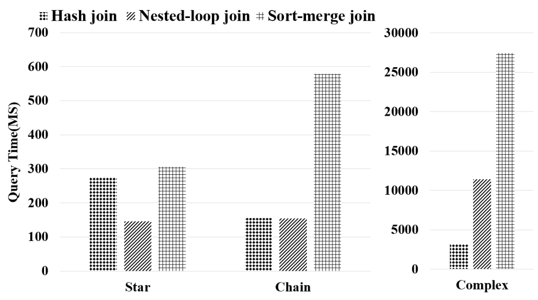
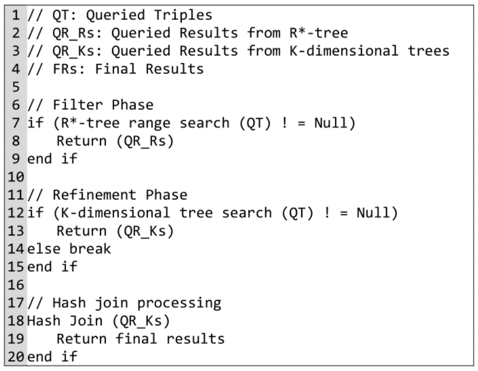


Fig. 4 Comparisons of three join algorithms

As shown in figure 4, the hash join algorithm has the most excellent performance among the three join algorithms through query performance comparisons of the three join algorithms. The hash join algorithm is very stable in the overall performance comparisons. Figure 5 shows a detailed description of our hash join algorithm based on the separated filter and refinement index structure.



```
// QT: Queried Triples
// QR_Rs: Queried Results from R*-tree
// QR_Ks: Queried Results from K-dimensional trees
// FRs: Final Results

// Filter Phase
if (R*-tree range search (QT) ! = Null)
   Return (QR_Rs)
end if

// Refinement Phase
if (K-dimensional tree search (QT) ! = Null)
   Return (QR_Ks)
else break
end if

// Hash join processing
Hash Join (QR_Ks)
   Return final results
end if
```

Fig. 5 Join algorithm based on separated filter and refinement index structure

## Ⅳ. Performance Evaluation

### 4.1 Experimental Data

In the experiment, we compare our convergence method with some existing popular methods to show that our method can achieve excellent join query performance. The experiments compared our convergence method (here we call Convergence) with Quad[9] and Darq[10]. Our experimental environment uses a system with 8GB of memory and a 3.4-GHz frequency processor.

To ensure that our experimental results are more convincing, we download the LUBM dataset[19]. LUBM dataset contains 230,061 triples, 38,334 subjects, 17 predicates, and 29,635 objects. We use 12 different SPARQL queries provided by[20] and two chain SPARQL query types that we constructed based on the LUBM dataset. We use three SPARQL query types[21], namely star type, chain type, and complex type. We will use the hash join algorithm for these SPARQL queries.

### 4.2 Performance Comparison

Figure 6 shows the performance of the star type. We can find that the query performance of Darq is abysmal. The query time of Quad and Convergence is significantly shorter than Darq. Among them, the performance of Convergence is the best.

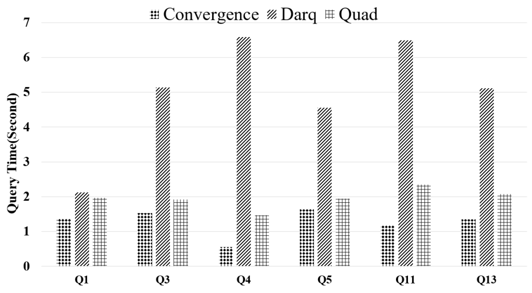


Fig. 6 Join query performance for star type

Figure 7 shows the performance of the chain type. The performance of Darq is still the worst. Quad saves nearly half of the time compared to Darq, and the performance of Convergence is better than Quad. Among the three methods, the query time of Convergence is the shortest.

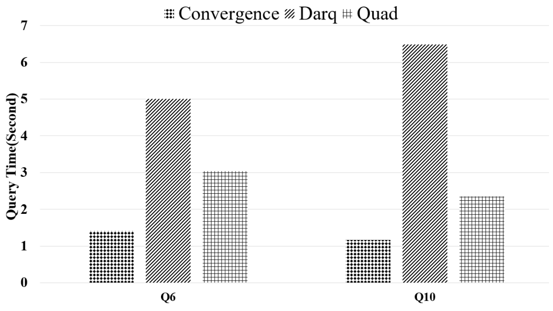


Fig. 7 Join query performance for chain type

Figure 8 shows the performance of the complex type. We find that the join query performance of Convergence is the best. Compared to Convergence, the query performance of Darq and Quad is not good.

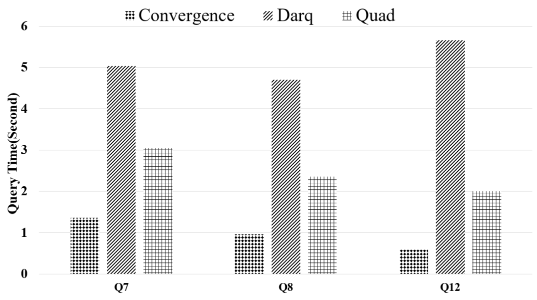


Fig. 8 Join query performance for complex type

Through the above experimental results, we observe that the performance of Darq is always the worst. Quad has consistently been in the middle of the three, and its query time is much faster than Darq. However, the performance of our Convergence method is better than Quad. In general, the query performance of our Convergence method is always better than other methods.

## Ⅴ. Conclusions

In this paper, we propose a convergence indexing method. This method consists of separated filter and refinement index structures with the hybrid storage system. The proposed index structure consists of an $R^*$-tree and K-dimensional trees. This convergence indexing method aims to filter unnecessary data and further refine the immediate result to improve the join query performance. In future work, we will continue to improve the performance of join queries by adopting different join algorithms and index structures.

Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. 2016R1D1A1B02008553).

## References

[1] M. Poblet, P. Casannovas, and V. Rodriguez-Doncel, *Linked Democracy: Foundations, Tools, and Applications*, Springer, 2019, pp. 1-25.

[2] H. S. Seok and Y. J. Lee, "Ontology-based IoT Context Information Modeling and Semantic-based IoT Mashup Services Implementation," *J. of the Korea Institute of Electronic Communication Science,* vol. 14, no. 4, 2019, pp. 71-76.

[3] M. Svoboda, "Efficient querying of distributed Linked Data," In *Proc.* 2011 *Joint EDBT/ICDT PhD Workshop,* Uppsala Sweden, 2011, pp. 45-50.

[4] A. Harth, K. Hose, M. Karnstedt, A. Polleres, K. U. Satler, and J. Umbrich, "Data summaries for on-demand queries over Linked Data," In *Proc. 19th International Conference on World Wide Web (WWW),* Raleigh, North Carolina, USA, Apr. 2010, pp. 411-420.

[5] G. Swathi, S. M. Hussain, P. Kanakam, and D. Suryanarayana, "SPARQL for semantic information retrieval from RDF knowledge base," *Int. J. of Engineering Trends and Technology (IJETT),* vol. 41, no. 7, 2016, pp. 351-354.

[6] O. Hartig, "An overview on execution strategies for Linked Data queries," *Datenbank Spektrum,* vol. 13, issue 2, 2013, pp. 89-99.

[7] C. Weiss, P. Karras, and A. Bernstein, "Hexastore: sextuple indexing for Semantic Web data management," In *Proc. Very Large Data Base (VLDB) Endowment,* vol. 1, no. 1, 2008, pp. 1008-1019.

[8] T. Neumann and G. Weikum, "The RDF-3X engine for scalable management of RDF data," In *Proc. Very Large Data Base (VLDB) Endowment,* vol. 19, no. 1, 2010, pp. 91-113.

[9] Y. X. Sun and Y. J. L, "Storage and Retrieval Architecture based on Key-Value Solid State Device," *J. of the Korea Institute of Electronic Communication Science,* vol. 15, no. 1, 2020, pp. 24-52.

[10] B. Quilitz and U. Leser, “Querying distributed RDF data sources with SPARQL”, In *Proc. 5th European Semantic Web Conf. (ESWC)*, Tenerife, Canary Islands, Spain, June 2008, pp. 524-538.
[11] M. Priti and H. E. Margaret, “Join processing in relational databases,” *ACM Computing Surveys*, vol. 24, no. 1, 1992, pp. 63-113.
[12] N. Beckmann, H. P. Kriegel, R. Schneider, and B. K. Seeger, “The R*-tree: An efficient and robust access method for points and rectangles,” In *Proc. ACM SIGMOD International Conference on Management of Data*, Atlantic City New Jersey, USA, 1990, pp. 322-331.
[13] A, Guttman, “R-trees: A dynamic index structure for spatial searching,” In *Proc. ACM International Conference on Management of Data*, vol. 14, no. 2, 1984, pp. 47 - 57.
[14] J. D. David and H. G.Robert, "Multiprocessor hash-based join algorithms," In *Proc. 11th international conference on Very Large Data Bases*, Stockholm, Sweden, 1985.
[15] T. David, C. H. C. Leung, W. Rahayu, and S. Goel, *High-Performance Parallel Database Processing and Grid Databases*, New York, Wiley, 2008.
[16] J. Gray, “What next? A dozen information technology research goals ACM turing award lecture,” *J. of the ACM*, vol. 50, no. 1, 2003, pp. 41 - 57.
[17] S. Nedev and V. Kamenov, “HDD performance research”, In *Proc. 8th International Scientific Conference Computer Science*, Greece, Kavala, 2018, pp. 106-111.
[18] N. Agrawal, V. Prabhakaran, and T. Wobber, “Design tradeoffs for SSD performance,” *USENIX Annual Technical Conf.*, Boston, Massachusetts, USA, June 2008, pp. 57-70.
[19] Y. Guo, Z. Pan, and J. Heflin, “An Evaluation of Knowledge Base Systems for Large OWL Datasets”, In *Proc. 3rd International Semantic Web Conference*, Hiroshima, Japan, 2004, pp. 274-288.
[20] C. R. Aberger, S. Tu, K. Olukotun, and C. Re, “Old techniques for new join algorithms: A case study in RDF processing,” In *Proc. IEEE 32nd International Conference on Data Engineering Workshops*, Helsinki, Finland, 2016, pp. 97-102.
[21] K. Lee and L. Liu, “Scaling queries over big RDF graphs with semantic hash partitioning,” In *Proc. Very Large Data Base (VLDB) Endowment*, vol. 6, no. 14, 2013, pp. 1894-1905.

## 저자 소개

**짜오티엔이(Tianyi Zhao)**

2019년 경북대학교 컴퓨터학과 졸업(공학학사)
2019년 ~ 현재 경북대학교 대학원 컴퓨터학과(공학석사)
※ 관심분야 : 시맨틱 웹, 빅데이터, 데이터베이스시스템

**이용주(Yong-Ju Lee)**

1985년 한국과학기술원 정보검색전공(공학석사)
1997년 한국과학기술원 컴퓨터공학전공(공학박사)
1998년 8월 ~ 현재 경북대학교 IT대학 컴퓨터학부 교수
※ 관심분야 : 링크드 데이터, 시맨틱 웹, 빅데이터, 지식 그래프