Regular paper

Journal of the KIECS. pp. 45-52, vol. 15, no. 1, Feb. 29. 2020, t. 99, pISSN 1975-8170 | eISSN 2288-2189
http://dx.doi.org/10.13067/JKIECS.2020.15.1.45

Key-Value Solid State Device 기반의 저장 및 검색 아키텍처

순위샹* · 이용주**

Storage and Retrieval Architecture based on Key-Value Solid State Device

Sun Yu Xiang* · Yong-Ju Lee**

요 약

본 논문에서는 저장 및 검색 성능과 보안을 고려하여 key-value 형태의 SSD를 활용한 RDF 데이터 저장 및 검색 문제에 대한 해결책을 제안한다. Key-value SSD를 사용한 RDF 데이터 셋으로 부터 논리 관계와 실제 값을 분리하기 위한 2단계 압축 알고리즘을 제안한다. 이는 압축 및 저장 성능뿐만 아니라 보안도 향상시킨다. 우리는 또한 검색 성능 향상과 병합정렬 조인 알고리즘 구현을 위한 R*-tree 기반 하이브리드 검색 구조를 제안했으며, R*-tree 검색 효율성에 영향을 미치는 요인들에 대해 설명한다. 논문에서 제안된 방식은 기존의 압축 및 저장 그리고 검색 접근 방식보다 저장 공간을 적게 차지하면서 더 빠른 결과를 얻을 수 있으며, 다양성, 유연성, 그리고 보안 측면에서 더 우수한 경쟁력을 가진다.

ABSTRACT

This paper proposes a solution for storage and retrieval problems for Resource Description Framework (RDF) data utilizing a key-value Solid State Device (SSD), considering storage, retrieval performance, and security. We propose a two-step compression algorithm to separate logical relationship and true values from RDF data-sets using the key-value SSD. This improves not only compression and storage efficiency but also storage security. We also propose a hybrid retrieval structure based on R*-tree to enhance retrieval efficiency and implement a sort-merge join algorithm, and discuss factors affecting R*-tree retrieval efficiency. Finally, we show the proposed approach is superior to current compression, storage, and retrieval approaches, obtaining target results faster while requiring less space, and competitive in terms of versatility, flexibility and security.

키워드

Storage and Retrieval Architecture, Key-value SSD, Sort-merge Join Algorithm, RDF, R*-tree, Linked Data.
저장 및 검색 아키텍처, Key-value SSD, 병합 정렬 조인 알고리즘, RDF, R*-tree, 링크드 데이터

Ⅰ. Introduction

Semantic web developments provide valuable information for new technology emergence, including big data, artificial intelligence and cloud computing, since the linked data concept is widely

* 경북대학교 IT대학 컴퓨터학부(syx@knu.ac.kr)
** 교신저자 : 경북대학교 IT대학 컴퓨터학부
· 접 수 일 : 2019. 10. 29
· 수정완료일 : 2019. 12. 22
· 게재확정일 : 2020. 02. 15
· Received : Oct. 29, 2019, Revised : Dec. 22, 2019, Accepted : Feb. 15, 2020
· Corresponding Author : Yong-Ju Lee
School of Computer Science and Engineering, Kyungook National University,
Email : yongju@knu.ac.kr

accepted [1]. This helps people to publish, share, and connect different data, information, or knowledge using a uniform format. The RDF data format is widely used to describe relationships between data and includes three features:

- RDF provides a logical data model for linked data and graph-based data formats.
- The core RDF feature is a triple that includes subject, predicate and object.
- The RDF data format is designed to help describe relationships between resources.

Thus, RDF forms a standard storage language, which offers several advantages:

- Different datasets can be easily connected by adding additional triples.
- RDF is much simpler than a relational database. Relational databases require designed tables with specific foreign keys to connect different tables.
- Since RDF is a graph based logical model, it offers great flexibility to express any data type.
- Valuable information and relationships can be extracted and discovered more easily from RDF based knowledge graph.

These advantages mentioned are also conductive to semantic web development. Therefore, the number of Linked Open Data (LOD) cloud datasets [2] has increased almost 100-fold in the past decade. Although dataset growth is conducive to the semantic web, several problems have arisen, including how to store and retrieve RDF data efficiently and securely. The storage bottleneck is determined by the data structure and storage hardware. Therefore, we propose a novel compression scheme that separates logical relationships from the data. We proposed a similar solution previously [3, 4], but subsequently found some weaknesses that required further optimized storage structure and algorithms to achieve efficient secure storage. The bottleneck for retrieving large RDF datasets is performing joins and unions, which is also an important big data research focus. To improve retrieval performance, we propose a hybrid indexing structure based on R*-tree, with a corresponding sort-merge join algorithm. We compare in compression, storage and retrieval performance for the proposed approach with current mainstream approaches. SSD use only a block interface. Therefore, data must be converted to allow applications to talk to virtually any solid-state drive. However, effective data conversion is costly from an operational standpoint and often become a performance bottleneck in scale-out and scale-up infrastructures. Samsung has developed a unique solution to this costly process that combines conventional SSD technology and the conversion layer into a single SSD [5]. The key-value SSD device not only simplifies the conversion process, but also significantly extends the drive's capabilities. The advantages mentioned above fit our storage requirements, and we will descript it in detail in section Ⅲ.

The remainder of this paper is organized as follows. Section Ⅱ describes related work. Section Ⅲ introduces storage and retrieval architecture based on a key-value SSD. Section Ⅳ provides performance comparison and analysis. Section Ⅴ summarizes and concludes the paper.

Ⅱ. Related Work

2.1 Compression methods

Compression methods can be divided into two general categories depending on whether the original data structure is changed or not. Direct compression is the most typical method and does not change the original data structure, whereas compression using linked lists, dictionaries, and cloud structures change the original data structure. We have discussed these methods previously [3], and in this section we just summarize previous

studies deficiencies and introduce a new compression method based on a key-value SSD.

Direct Compression: Direct Compression (DC) is one of the most widely used compression methods because it is simple and does not change the original data structure. Compression efficiency depends on the specific algorithm, but it is difficult to obtain satisfactory performance , which leads to various practical application limitations.

Adjacency Lists: Section 4.1 discusses RDF data characteristics in detail, and one of the most important characteristics is the large number of duplicates items. Thus, Adjacency Lists (ALs) are commonly adopted to reduce duplication, representing triples as Subject → [(Predicate$_1$, ObjList$_1$), (P_K, ObjList$_K$)].

Dictionary + Triples: This approach replaces dictionary format triples true values with hash values, such as <HashKey, TrueValue>. One advantage of the Dictionary + Triples (DT) approach is high flexibility, logical relationship can be changed by changing the dictionary mapping relationship. However, a serious disadvantage is low compression efficiency.

Cloud Compression: Cloud compression (CC) offers excellent compression performance by separating logical relationships and true values from the data, and then utilize cloud to manage true values. This improves not only compression and storage efficiency but also storage security.

DT compression is the most promising method even though compression efficiency is low, because other methods offer unsatisfactory flexibility and extensibility. Therefore, we adopt a key-value SSD storage method to address low compression efficiency when storing the dictionary. This storage mode logical relationship and true values from RDF data, and not only greatly improves storage performance but also security. The proposed key-value SSD storage method offers several advantages over private cloud platform compression: it provides significant I/O performance improvement compared with traditional relational databases; and its simpler structure is easier to extend in hardware. The key-value SSD device adopt RocksDB as embedded NoSQL database for persistent key-value storage, and using log structured merge tree architecture. It originated by Facebook and actively used in their infrastructure.

2.2 Storage systems

Much previous research has focused on RDF query processing and storage structure, which can be classified into centralized and distributed storage. Centralized storage systems are very popular for semantic based application development. People can obtain metadata from different datasets, reprocess it to produce new valuable information, and store it locally. The biggest advantage is high retrieval performance and utilizing various retrieval structures flexibly, including B+ trees [6] and path indexing [7], to improve retrieval efficiency [8, 9]. However, there is a fatal drawback that local storage information cannot be guaranteed up to date.

Distributed storage can address centralized storage issues, and in contrast does not need to preprocess metadata since endpoint data is queried and processed only when a request is received. Although distributed storage guarantees data reliability, data security and retrieval performance are challenging, because all processing must use the network [10 - 12].

III. Storage and Retrieval Architecture based on Key-Value SSD

Figure 1 shows that the proposed systems architecture includes five modules: Data Filtering Module (DFM), Parsing Module (PM), Compression Module (CM), Transport Module (TM), and

Retrieval Module (RM). White regions represent modules, light gray represents functions, dark gray represents key-value SSD, light blue represents dictionary, and yellow represented clients.

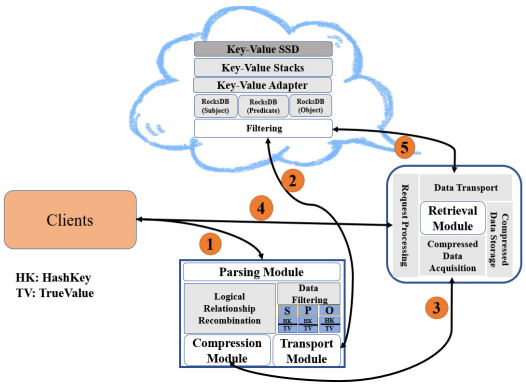


Fig. 1 Key-value SSD storage and retrieval architecture

- *Data Filtering Module*: The DFM has two main functions: filter converted data and store them separately in RocksDB (Fig. 1., line 2); and convert requested data from RM module to true values using hash keys (Fig. 1., line 5).
- *Parsing Module*: The main PM function is to accept data from CM and parse it into dictionaries and logical relationships. The dictionaries become <HashKey, TrueValue>, and an adjacency list is generated based on hash values without breaking the original logical relationship.
- *Compression Module*: The main CM purpose is to compress the recombined logical relationship dataset and send it to RM (Fig. 1., line 3).
- *Transport Module*: The main TM task is to transfer converted dictionary information to the cloud (Fig. 1., line 2).
- *Retrieval Module*: The RM is core architecture module. RM processes client requests (Fig. 1., line 4) and response results. It acquires and initializes data from CM.

The five modules constitute the architecture ontology. Functionally, the architecture comprises three services: lines 1 and 2 provide data parsing; lines 1 and 3 provide data compression; and lines 4 and 5 provide data retrieval.

3.1 Two-step compression algorithm

Figure 2 shows proposed two-step compression algorithm. This section introduces a two-step compression algorithm that differ somewhat from our previous algorithm [3], particularly with regard to logical relationship recombination. The adjacency list starting point is replaced by items with highest repetition. Section Ⅳ shows that the number of objects is larger than subjects or predicates. Therefore, we can omit calculating repetition rate for each triple item.

ALGORITHM 1: Two-step Compression Algorithm

```
Model ← Create Default Model
Model.read(Dataset_file)
iter ← Model.listStatements()
AdjLists ← HashMap<Integer, HashMap<Integer, Set<Integer>>>()
while (iter.hashNext()), do
        stmt ← iter.nextStatement()
        subject ← stmt.getSubject().hashCode()
        predicate ← stmt.getPredicate().hashCode()
        object ← stmt.getObject().hashCode()
        if (AdjLists.get(predicate) is null), do
                s ← HashSet<Integer>()
                s.add(object)
                ms ← HashMap<Integer, Set<Integer>>()
                ms.put(subject, s)
                AdjLists.put(predicate, ms)
        end
        if (AdjLists.get(predicate) is non-null and AdjLists.get(predicate).get(subject) is null), do
                s ← HashSet<Integer>()
                s.add(object)
                AdjLists.get(predicate).put(subject, s)
        end
        if (AdjLists.get(predicate) is non-null and AdjLists.get(predicate).get(subject) is non-null), do
                AdjLists.get(predicate).get(subject).add(object)
        end
end
ppmdi (AdjLists.JSON)
```

Fig. 2 Two-step compression algorithm

The proposed two-step compression algorithm creates a default model and loads the dataset, creates an adjacency list based on HashMap, and recombines logical relationships by iterating triples.

This algorithm then converts the adjacency list to JSON and compresses it using ppmdi, which offers higher compression efficiency compared with the previously adopted gzip.

3.2 Hybrid retrieval structure based on R*-tree

R*-tree is a 3D spatial index structure commonly used as to index geographic information. Although R*-tree has slightly higher construction cost than R-tree, it offers better query performance. The critical factor affecting retrieval performance is bucket size, with retrieval performance inversely proportional to bucket size. Therefore, bucket size and structure are key to improve query performance. We manage bucket data using linked lists and sorted lists, with corresponding O(n) and O(log n) time complexities. In principle, sorted list advantages will become more significant with increased data size, and sorted lists are a prerequisite for the sort-merge join search. Therefore, we adopted sorted lists and the sort-merge join algorithm. Figure 3 shows the proposed hybrid retrieval structure based on R*-tree.

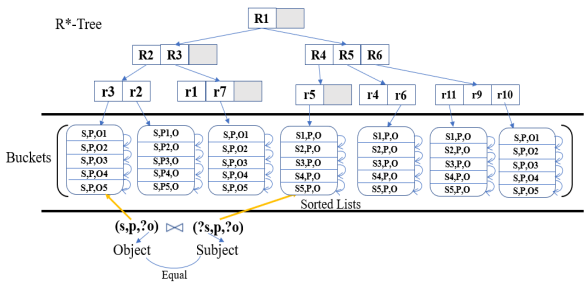


Fig. 3 Hybrid retrieval structure based on R*-tree

The entire retrieval was divided into R*-tree and buckets, where the R*-tree task is to manage buckets in range, and then sort the bucket data according to the queried items of triples. For example, suppose the triple (s, p, ?o) needs to be retrieved. The bucket data is sorted according to object size. We use the sort-merge join algorithm to query the final results.

ALGORITHM 2: Proposed Sort-Merge Join Algorithm

```
if (Queried triples in R*-tree), do
        #Assume that all buckets are already sorted and non-null
        while (left_bucket has next and right_bucket has next), do
                if (left_bucket.key equal right_bucket.key is true and other terms satisfy the condition), do
                        results.add(right_bucket.key);
                        left_bucket.Advance();
                        right_bucket.Advance();
                end
                if (left_bucket.key less than right_bucket.key is true), do
                        left_bucket.Advance();
                end
                if (left_bucket.key greater than right_bucket.key is true), do
                        right_bucket.Advance();
                end
        end
end
```

Fig. 4 Sort-merge join algorithm

Figure 4 shows the proposed sort-merge join algorithm based on the hybrid retrieval structure. We assume that all buckets are already sorted and non-null, and compare the sorted bucket data using hash value size. If equal, we compare the remaining known terms. The major advantage from this algorithm is stable performance. Compare to nested join search with O(m*n) time complexity, the average time complexity for the proposed sort-merge join search is O(n*log(n)+m*log(m)).

Ⅳ. Performance Comparison and Analysis

This section compares the proposed approach with current mainstream compression and storage methods. To provide realistic tests, we downloaded datasets from four different fields. DBpedia [13], contains RDF information extracted from Wikipedia. DrugBank [14] is a unique bio-informatics and chem-informatics resource that combines detailed drug data with comprehensive drug target information. LinkedGeoData [15] is a large spatial knowledge base derived from OpenStreetMap for

the semantic web. Images [16] contain image information extracts from DBpedia.

4.1 Characteristics of experimental data

Since RDF data structure can be easily converted to points in 3D space using a hash function (e.g., hashcode in java). We created visual representations for the real datasets in 3D space, as shown in Figure 5. There are several common points among the selected datasets, summarized below and details in Table 1.

- They are significantly stratified and perpendicular to the property axis, particularly for the images dataset, hence there are many duplicate predicates.
- Data is closely scattered within each layer, i.e., the repetition rate for subject and object is irregular.
- Data distributions are relatively concentrated, with relatively rare special data occurrences.

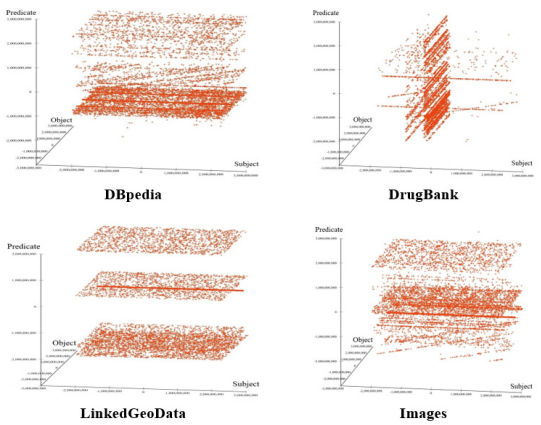


Fig. 5 Dataset visualizations

4.2 Compression performance

To verify join query accuracy, we randomly inserted two sets of triples that meet join query condition into three datasets and compare their join query time by querying the same data. The operating system was Windows 10 ® Professional, with 8 GB RAM and 3.60 GHz CPU. Table 2 shows compression rates for different approaches.

Table 1. Experimental dataset characteristics

Dataset	Size(MB)	No. Triples	No. Subjects	No. predicates	No. Objects
DBpedia	3.94	31,050	4,008	23	16,644
DrugBank	144	766,920	19,693	119	274,864
LinkedGeoData	327	2,207,295	552,541	1,320	1,017,242
Images	2,037	10,108,218	4,365,622	5	4,130,923

Table 2. Compression rate

Dataset	DC	AL	DT	CC	TC
DBpedia	9.37%	7.49%	90.76%	4.29%	4.04%
DrugBank	7.72%	6.72%	25.55%	1.91%	1.42%
LinkedGeoData	6.26%	6.26%	49.46%	3.63%	3.01%
Images	5%	11.21%	48.57%	4.8%	3.8%

TC performance is slightly superior to CC and the test results confirm the proposed two-step algorithm achieves suitable effects. More importantly, dataset statistics and analysis provided

some general data distribution rules, which greatly help for logical relationship recombination.

4.3 Storage performance

Figure 6 compares storage performance among the considered approaches. The proposed separated storage (SS) approach achieved superior storage efficiency. Yars [6] centralized storage approach stores RDF data using six B+ tree indices, where the key for each B+ tree is a concatenation of the subject, predicate, object, and context. The six indices cover all the possible access patterns. Therefore, Yars requires significantly more storage space compared with other storage approaches. Midas [10] and Darq [11] are distributed storage approaches, with Midas using an index structure based on K-dimensionality tree.

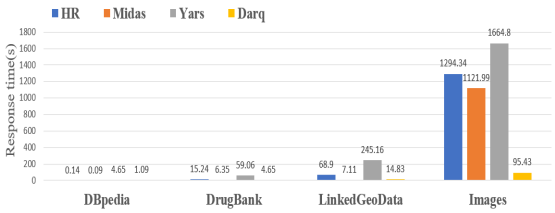


Fig. 6 Storage performance

4.4 Retrieval performance

Figure 7 compares retrieval performance for the considered approaches. The proposed hybrid retrieval (HR) approach offers significant advantages. The excellent spatial index structure from R*-tree and efficient join algorithm are critical components. Midas has inferior performance compared with HR, with the performance gap between K-dimensionality tree and R*-tree index approaches increasing with increasing data volume. Yars approach has similar significant performance attenuation, suggesting B+ tree does not perform better for large multi-dimensional data retrieval.

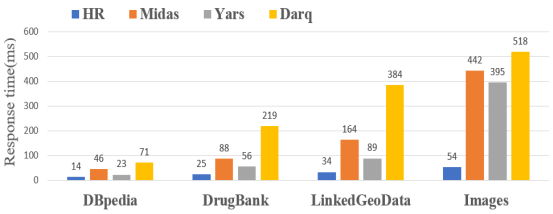


Fig. 7 Retrieval performance

Ⅴ. Conclusions

Efficient storage and retrieval are fundamental requirements for big data, machine learning and other emerging technologies. We propose a novel solution to efficiently and securely store and retrieve 3D data utilizing key-value SSD. The proposed storage strategy separates true values and logical relationships from datasets, providing enhanced performance and security, and guaranteeing data security in the retrieval process. With the development of 5G networks, terminal information storage based on key-value SSD will become the mainstream multi-dimensional data storage. In future research, this storage and retrieval solution will be evaluated in real semantic-based application, such as [17, 18].

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] C. Bizer, T. Heath, K. Idehen, and T. Berner-Lee, "Linked Data on the Web," In *Proc. World Wide Web*, Beijing, China Apr. 2008, pp. 1265-1266.

[2] LOD (Linked Open Data) Datasets, 2019.

[3] Y. X. Sun, S. H. Lee, and Y. J. Lee, "Cloud Storage Platform for Efficient RDF

Compression," In *Proc. 11th International Conference on Computer and Electrical Engineering*, Tokyo, Japan, Oct. 2018, pp. 1-5.

[4] 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. 1990 ACM SIGMOD international conference on Management of data (SIGMOD)*, Atlantic City, New Jersey, USA, 1990, pp. 322-331.

[5] Samsung Key Value (KV) SSD Technology Brief, 2017.

[6] A. Harth and S. Decker, "Optimized Index Structures for Querying RDF from the Web," In *Proc. 3rd Latin American Web Congress (LA-Web)*, Buenos Aires, Argentina, Oct. 2005, pp. 71-81.

[7] B. Liu and B. Hu. "Path Queries based RDF Index," In *Proc. 1st International Conference on Semantics, Knowledge and Grid*, Beijing, China, Nov. 2005, pp. 91-93.

[8] C. Wess, P. Karras, and A. Bernstein, "Hexastore: Sextuple Indexing for Semantic Web Data Management," In *Proc. 34th International Conference on Very Large Data Bases (VLDB)*, Auckland, New Zealand, Aug. 2008, pp. 1008-1019.

[9] T. Neumann and G. Weikum, "RDF-3X: A RISC-style Engine for RDF," In *Proc. 34th International Conference on Very Large Data Bases (VLDB)*, Auckland, New Zealand, Aug. 2008, pp. 647-659.

[10] G. Tsatsanifos, D. Sacharidis, and T. Sellis, "On Enhancing Scalability for Distributed RDF/S stores," In *Proc. 14th International Conference on Extending Database Technology (EDBT)*, Uppsala, Sweden, 2011, pp. 141-152.

[11] B. Quilitz and U. Leser, "Querying Distributed RDF Data Sources with SPARQL," In *Proc. 5th European Semantic Web Conf. (ESWC)*, Tenerife, Canary Islands, Spain, 2008, pp. 524-538.

[12] A. Langegger, W. Wob, and M. Blochl, "A Semantic Middleware for Virtual Data Integration on the Web," In *Proc. 5th European Semantic Web Conference (ESWC)*, Tenerife, Canary Islands, Spain, 2008, pp. 493 - 507.

[13] DBpedia, Provided by Wikipedia, 2019.

[14] DrugBank, Supported by the Canadian Institutes of Health Research, 2019.

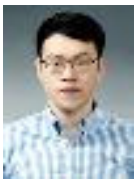
[15] LinkedGeoData, Administered by the AKSW research group, 2019.

[16] Images, Download from Wikipedia, 2019.

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

[18] C. W Kim and J. W Kim, "Image Retrieval System of semantic Inference using Objects in Images," *J. of the Korea Institute of Electronic Communication Sciences*, vol. 11, no. 7, 2019, pp. 677-684.

저자 소개



순위샹(Sun Yu Xiang)

2016년 안동대학교 컴퓨터학과 졸업(공학학사)
2019년 경북대학교 대학원 컴퓨터학과 졸업(공학석사)
2019년 ~ 현재 경북대학교 대학원 컴퓨터학과(공학박사)
※ 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,웹 사이언스