Linked Data Processing on Embedded Devices

Anh Le-Tuan, Conor Hayes
Insight Centre for Data Analytics, National University of Ireland Galway
E-mail: {anh.letuan, conor.hayes}@insight-centre.org

Abstract

The constant improvement of computing capability and network connectivity on embedded devices has enhanced the development of ubiquitous systems and the Internet of Things. Meanwhile, the Semantic technologies are the key enabler for such developments. There have been evidences of the limitations of centralised paradigm when the number of devices is explored [9, 7]. Thus, distribute semantic data processing locally on devices is emergence. In this paper, we firstly address the challenges while process Linked Data on embedded devices. Next, we propose an approach of adapting database management that scales such processing.

1. Introduction

Pervasive computing and the Internet of Things are facing the challenges on representing, integrating and reasoning on sensors' data. Linked Data provides a promising solution to these difficulties. RDF is a well established data model to describe the semantics of real data [2]. As well as allowing a flexible way of integrating heterogeneous data, and RDF ontology-based context description enables better reasoning and a better sharing contextual information [8].

Semantic processing can be applied remotely and devices can act as an interface to semantic data without having to handle it directly. Executing the data processing tasks locally on embedded devices might require much more efforts in optimising the computations or in handling limited resources. However, devices can be self-contained and be able to operate in different environments. Furthermore, data transmission costs can be dramatically reduced as it does not require the transfer of data from a device to a server. Working independently from a remote server also avoids the requirement for devices to maintain a frequent connection. Thus, the risks caused by intermittent connectivity can be reduced. As device data is not stored and processed on a remote server, the privacy and security concern is also reduced. Finally, by distributing that computation among a large number of existing devices, a greater computational scale can be achieved.

In this paper, we address the performance issues that existing RDF frameworks suffer while running on embedded devices. To overcome these issues, we then propose an approach of using database management techniques.

2. Research Objectives

Embedded devices are memory-constrained. If main memory is used without awareness of the limitation, embedded systems face high risk of crashing due to out-of-memory. The existing implementations, such as Sesame or Jena, tend to use GBs main memory as high speed buffer for data processing. Our experiments show that Sesame, Jena and its ported version for AndoJena suffer in scalability issues due the memory limitation of devices.

Since memory is limited on embedded devices, and out-of-memory errors crash applications, data may be frequently written and read from secondary storage. Embedded devices use flash memory as secondary storage. Data structures and indexing schemes for traditional magnetic disk work inefficiently on flash-based storage due to the differences in physical data management between these two types of memory medium [1, 5]. As in the initial version of RDF on the go [6], Berkeley DB could adapt to the low memory environment, however, it writes and read RDF data very slowly. Hence, The second objective is how RDF data can be organised so that it may be efficiently accessed on flash-based storage of embedded devices. To the best of our knowledge, this question currently has no attention.

3. Approach

In this section, we propose our approach to overcome the issues that were addressed in previous section. To reduce memory consumption of RDF data, we apply RDF encoding approach. In flash memory, bits are organised as fixed size blocks. The I/O unit on flash-based storage is block based. The size of I/O equals to the block size of flash memory. Flash memory have no mechanism seek latency. However, flash memory suffers erase-before-write limitation that causes its poor performance with random writes. Therefore, we introduce an alternative physical organisation for RDF data called the Two-layers indexing that could reduce the number of random writes. Finally, we introduce resilient buffer manager that could adapt to availability of memory.

3.1 RDF Encoding

RDF encoding is a compression technique that is commonly used in many triple stores[10, 4]. An Encoder, a Decoder along with a Dictionary are responsible for compressing and decompressing RDF nodes. When most of the operations of RDF nodes, such as matching during a query execution, can be performed in the compressed form. Only encoded RDF Nodes are cached in main while their string representations are kept on flash storage. When it is needed, the Decoder will return the original form.

3.2 Physical Organisation

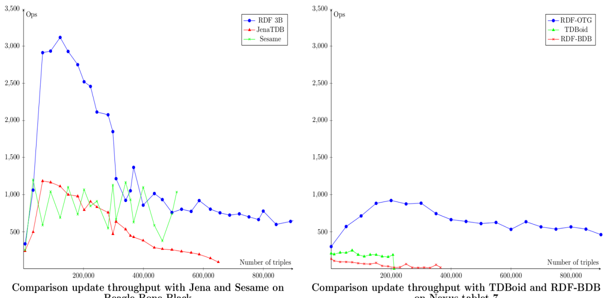
In a file, RDF triples are encoded and organised into three components tuples. The tuples are sorted lexicographically and compressed into fixed-size blocks. The size of block equal flash devices erase block. A spare index holds the positions of data blocks is small enough to fit into main memory. Each block is identified by its lowest tuple and its highest tuple. Free space is always left in each data block to insert new tuples. Thus, it only has to update the modified data block instead of resorting the whole file that may require many rewriting operations. When a block is split for new space, the updated part will be copied into a new block and is assigned as the last data block of the file. The old block remains the same, but the sparse index is updated with its new lowest tuple and highest tuple. This block is updated later when a new tuple arrives.

3.3 Buffer Management

Data is always read and written from flash-based storage in fixed size memory blocks. Hence, the buffer should organise data in memory blocks of the same size. To avoid out-of-memory errors, the buffer manager writes data to storage to claim free memory. The storage I/O is much slower than other operations in the system. Therefore, the number of reads and writes must be reduced as much as possible. In buffer, a score is assigned to each block to detect its access frequency. From the score, the buffer manager chooses and writes the block with the lowest score to the storage and keeps the more active block in main memory.

4 Preliminary Results

To test our approach, we compare the scalability of our system to other systems that could run on the same devices. The scalability means that the size of RDF dataset that the system can support and answer queries in an acceptable delay. We use generated data and SPARQL queries from BSBM benchmark [3] to test the RDF storage and the SPARQL processor. On a BeagleBone Black, we compare our implementation with Jena and Sesame. On an Android tablet Nexus 7, we compare our RDF-OTG with AndroJena and RDF-BDB, which uses Berkeley DB for RDF storage.

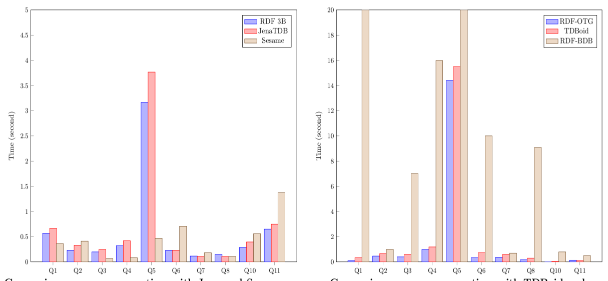


Comparison update throughput with Jena and Sesame on Beagle Bone Black

Comparison update throughput with TDBoid and RDF-BDB on Nexus tablet 7

Figure 1: Update Throughput

The throughput tests results are illustrated in Figure 1. On the Beagle Bone, the native store of Sesame and JenaTDB can load 500k triples and 600k triples. The tests stopped due to out-of-memory errors. Our engine, RDF 3B, can insert more than 1 million triples with higher throughput. On the tablet, our system RDF-OTG also shows greater efficiency in inserting rate and scale. The directly ported version of Jena crashed after inserting 200k triples. RDF-BDB does not run out of memory, however, its throughput is very low (10-20 Ops). Hence, the B++ Tree index structure of Berkeley DB is not sufficient on flash memory.



Comparison query response time with Jena and Sesame on Beagle Bone Black

Comparison query response time with TDBoid and RDF-BDB on tablet Nexus 7

Figure 2: Query Response Time

The figure 3 reports the response time of SPARQL queries on respectively 500k triples on the BeagleBone and 200k on the tablet. Our framework can answer all SPARQL queries less than 4 seconds on the BeagleBone Black and 16 seconds on the tablet. However, as we reuse the SPARQL processor of Jena, our implementations perform slightly better than Jena because it can access RDF data on flash faster. The memory profiling shows that for the same size of data, our implementation consume one-third memory as Jena does and a half as much as Sesame requires.

5 Conclusion

In this paper, we presented an approach for improving performance of Linked Data processing on embedded devices. Our initial results promisingly show that database optimisations can support better scalability and performance for RDF data on embedded devices.

Acknowledgement: This thesis is funded by Irish Research Council under Grants No. GOIPG/2014/917.

References

[1] D. Ajwani, I. Malinger, U. Meyer, and S. Toledo. Characterizing the performance of flash memory storage devices and its impact on algorithm design. WEA, 2008.

[2] P. M. Barnaghi, W. Wang, C. A. Henson, and K. Taylor. Semantics for the internet of things: Early progress and back to the future. *Int. J. Semantic Web Inf. Syst.*, 8(1):1–21, 2012.

[3] C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. *International Journal on Semantic Web and Information Systems*, 2001.

[4] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame : A generic architecture for storing and querying rdf and rdf schema. *ISWC*, 2002.

[5] I. Koltsidas and S. D. Viglas. Data management over flash memory. *SIGMOD*, 2011.

[6] D. Le-phuoc, A. Le-tuan, G. Schiele, and M. Hauswirth. Querying Heterogeneous Personal Information on the Go. *ISWC*, 2014.

[7] W. Shi and S. Dustdar. The promise of edge computing. *Computer*, 49(5):78–81, May 2016.

[8] T. Strang and C. Linnhoff-Popien. A Context Modeling Survey. *UbiComp*, 2004.

[9] B. Varghese, N. Wang, D. S. Nikolopoulos, and R. Buyya. Feasibility of fog computing. *CoRR*, abs/1701.05451, 2017.

[10] K. Wilkinson, C. Sayers, H. Kuno, and D. Reynolds. Efficient RDF storage and retrieval in Jena2. *SWDB*, 2003.