Linked Data processing for Embedded Devices

Anh Le-Tuan

INSIGHT Centre for Data Analytics
National University of Ireland, Galway.

1 Problem Statement

An important goal of pervasive computing or the Internet of things(IOTs) is the enabling of smart services and applications that proactively adapt to the context of users and surrounding environments [20, 15]. In principle, these systems consist of embedded and networked computing devices that can automatically detect context and make an appropriate adaptation. However, allowing autonomous detect of context on the devices is difficult. Data from sensors should be collected to produce machine readable situational awareness data that allows for the intelligent response. There are many challenges in representing, integrating and reasoning on volatile and diverse data.

Linked Data provides a promising solution to these difficulties. RDF is a well established data model to describe the semantics of real data [3]. 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 [22].

While many embedded devices have enough computing resources and storage for processing RDF data locally, this is still a challenging task. The existing RDF libraries for embedded devices are limited in functionality, and the RDF frameworks for PC-based workstations suffer performance issues running on such devices. Our work aims to a comprehensive, scalable and resourced-awareness software framework to process RDF data for embedded devices.

In this thesis, we introduce a system architecture that support RDF storage, SPARQL Query, RDF reasoning and continuous query for RDF stream on embedded devices. To achieve the efficient performance and scalability, we propose data management techniques that adapt to hardware characteristics of embedded devices. Computing resources on embedded devices are constrained, their usage should be context-dependent. Therefore, we work on an adaptation model that supports trading off system performance and device resources depending on their availability. The model is based on the cost model of the data management techniques.

2 Relevancy

On a powerful server, Linked Data can be efficiently processed with existing RDF frameworks such as Sesame [5] or Jena [23]. In many context-aware systems [16], raw data from connected devices is sent to a server from which contextual information is created, stored and queried. However, due to the distributed characteristic of pervasive and IOT environments, the centralised architecture is a

weak point. In such environments, it is provisioned that devices, physical objects are autonomous, independent, interoperable and easily added or removed [19, 7].

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.

3 Related works

There have been several works that try to ship RDF data processing to embedded devices. Mobile RDF [1] is a lightweight RDF framework that is suitable for Java Me. This library allows simple APIs such as creating, parsing and serialising RDF data, but RDF graph modifications are not supported. Micro-Jena [2] is an early adoption of Jena to J2ME that is specifically developed for Symbian OS. However,it only provides in-memory processing. AndroJena [3] is another adoption of Jena that offers all functionalities as original Jena framework. However, ignoring the fact that RDF data processing cannot be directly applied to the mobile setting, Androjena has scalability issue due to memory limitation of mobile devices (as shown in our experiment). Wiselib tuplestore [8] is a notable work that is recently introduced. Attempting to tackle the resource constraint, the authors tried to reduce code size and build a database tailored to embedded devices. However, this library does not include SPARQL processor or inferencing engine.

In the response to these dependencies, an initial RDF storage solution and SPARQL query processor for android devices, RDF on the Go has been developed [14]. To overcome the memory limitation of Android, this work uses a lightweight version of Berkeley DB [4] to store data in the secondary storage. However, its performance is inefficient due to the slow read operation and write operation of Berkeley DB. In a better developed version [13], we provide an Android native RDF storage to adapt to the characteristic of Android devices.

4 Research questions

Considering the limitations identified in previous sections, the main goal of the research is to enable a scalable and resource-efficient framework for processing Linked Data on embedded devices. To achieve the research goal, we address the following research questions:

[1] http://www.hedenus.de/rdf/
[2] http://poseidon.ws.dei.polimi.it/ca/?page_id=59
[3] https://github.com/lencinhaus/androjena
[4] http://www.oracle.com/technetwork/ database/berkeleydb/overview/index.html

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 pc-based 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. Therefore, our first research question(**RQ1**) is how to enable a greater scale of RDF data for memory-constrained 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 [2, 10]. As in the initial version of RDF on the go [14], Berkeley DB could adapt to the low memory environment, however, it writes and read RDF data very slowly. Hence, the second research question(**RQ2**) 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.

Resource availability on embedded devices is hard to predict. For example, there may be several background services running to collect data, or connection bandwidth changes due to changing place. Therefore, resource consumption should be context dependent and may change according to need. For example, on battery-powered devices, optimising energy consumption to keep devices alive should be in a high priority. Hence, how to adapt the optimisations in (**RQ1**) and (**RQ2**) to available resources on embedded devices is consequently our third research question(**RQ3**).

5 Hypotheses

In this section, we present our provisional hypotheses. We expect that more will become apparent as this research matures:

H1: A great scale of RDF data can be achieved on embedded devices by compressing RDF nodes to reduce the memory consumption and use the secondary storage as extended buffer.

H2: RDF data can be read and written from/to store efficiently if its physical organisation is applicable to the I/O behaviours of physical storage [9, 18].

H3: Defining the cost models of each resources for the system, the resources consumption can be determined. According to the cost models, resources consumption can be traded-off depending on their availability.

6 Approach

Our research goal is to improve the scalability and the resources efficiency of RDF data processing for embedded devices. This concern is related to database management's system performance that is strongly impacted by the characteristic of hardware resources [1]. Our approach is to adapt embedded database management to manipulate RDF data. The next two subsections summarise

our system overview the adaptation and appropriate database techniques for embedded hardware.

6.1 System Overview

Follow the recommendations of designing embedded database [17], the system is designed as functionalities customisable. That means the applications can chose and include only the required features. To reduce the memory consumption of RDF data, we propose using RDF encoding(**H1**). This compression technique is commonly used in many triple stores [23, 5]. The system includes a shrinking buffer to claim free memory(**H1**) when need. In secondary storage, we use data structure and indexing techniques supporting high throughput to trade off the I/O asymmetry of flash-based storage(**H2**). To trade off resource consumption, optimising execution plans is offered by an *Execution Engine*(**H3**).

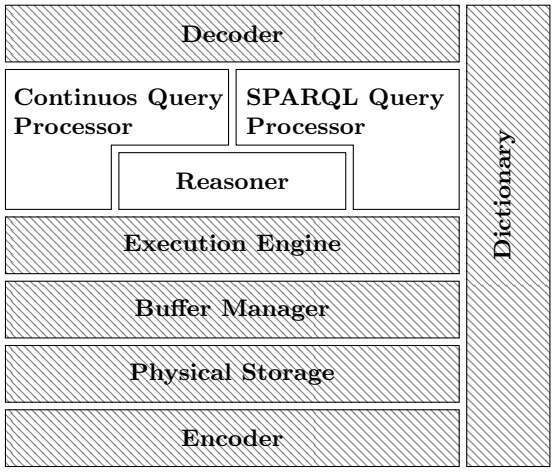


Fig. 1. Architecture

As illustrated in Figure 1, the overall architecture consists of building blocks, each block describes a system's component. An *Encoder*, a *Decoder* along with a *Dictionary* are responsible for compressing and decompressing RDF nodes. The Execution Engine employs execution plans and provides RDF graph APIs. So that, the functionalities components such as *Continuous Query Processor*, *SPARQL Query Processor*, *Reasoner* can manipulate RDF data through the standard RDF graph APIs. In the physical layer, a $Buffer\ Manager$ is tightly coupled with a *Physical Storage* to manage in-memory data and persistent data.

6.2 Adaptation techniques

The hardware characteristics of embedded devices that significantly influence systems performance of a database management system are the limitation of main memory and the novel I/O behaviours of flash-based storage. In following, we propose database techniques to adapt RDF processing data to such hardware environment(**RQ1, RQ2**).

Using RDF encoding, RDF nodes are transformed into 32-bit length integers. Most of the operations of RDF nodes, such as matching during a query execution,

can be performed in the compressed form. Only encoded integers are cached in main while their string representations are kept on flash storage. When it is needed, the Decoder will return the original form. To reduce space required to store the strings, prefix mapping techniques and Huffman coding [21] are also applied.

Data is read and written from flash-based storage in fixed size memory blocks. Hence, the Buffer Manager and the Physical Storage 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 the Buffer Manager, 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. To manage the intermediated data from joins, we also use indexes on bags of mapping [12] to optimise buffer size.

On flash-based storage, reading is much faster than writing because it has no mechanical seek latency. Overwriting in flash memory is significant slow due to the erase-before-write limitation [10]. To trade off flash I/O asymmetry, to fasten deallocating data, to support intensive inserting RDF triples, we try to optimise the write operation of the Physical Storage. Therefore, we use a two-layers index to manage persistent data. In a file, tuples are sorted lexicographically and are compressed into fixed-size blocks. The spare index holds the positions of data blocks in the system files, and it 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. It might waste space since data blocks in a file are not entirely full, however, this is a standard strategy (also on non-Flash storage) to trade space efficiency with performance.

7 Evaluation plan & Preliminary Results

To test our approaches, we compare the scalability of our system to other systems that could run on the same devices(**H1, H2**). The scalability means that the size of RDF dataset that the system can support and answer queries in an acceptable delay. Furthermore, we plan to simulate scenarios of changing resources to test the resource consumption adaptation model(**H3**).

The following is our experimental results of our current work's stage. The experiment evaluates the updating throughput and scalability of RDF storage, the response time of SPARQL queries processor and the ability of processing continuous queries. As we have found LUBM's dataset and queries are not applicable to evaluate reasoners on small devices [6], the evaluation on reasoner is currently future work.

To set up the evaluation, we implement a testing prototype in Java. On top of that, we integrate the SPARQL query processor of Jena [23] and the version for embedded devices of CQELS [11]. We use generated data and SPARQL queries from BSBM benchmark [4] to test the RDF storage and the SPARQL processor. The throughput test is a simulation of data growing process by gradually adding more data. The response time of queries is measured on the same dataset that all engines can support. On a BeagleBone Black [5], we compare our implementation RDF 3B with Jena and Sesame. And on an Android tablet Nexus 7 [6], we compare our RDF-OTG with AndroJena and RDF-BDB, which uses Berkeley DB for RDF storage.

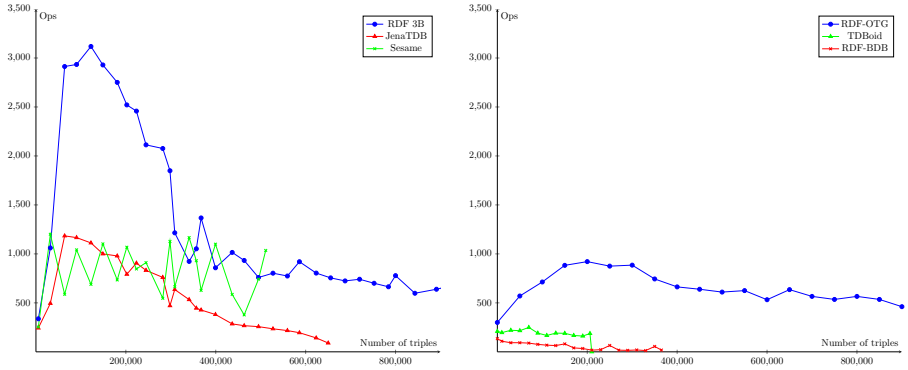


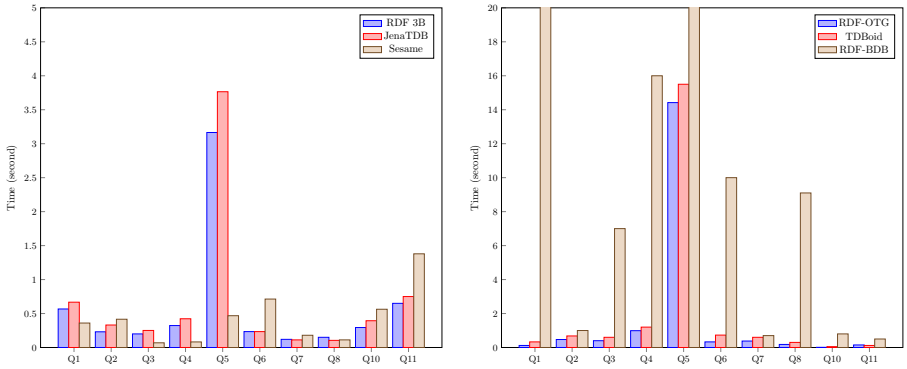
Fig. 2. Update throughput

The throughput tests results are illustrated in Figure 2 2. On the BeagleBone, 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. The update throughput reaches a peak of 3000 operations per second(Ops) and remains stable at 1000 Ops. On the tablet, our system RDF-OTG also shows greater efficiency in inserting rate and scale. The directly ported version of Jena, the TDBoid crashed after inserting 200k triples. With the shrinkable mechanism of Berkeley DB, RDF-BDB does not run out of memory. However, its inserting rate is very low (10-20 Ops). Thus, we show PC-based implementations work inefficiently on embedded devices due to the memory constraint. The B++ Tre indexing technique (of Berkeley BD) may work efficiently on magnetic disk, but it is inefficient for flash-based storage

The figure 3 reports the response time of SPARQL queries on respectively 500k triples on the BeagleBone and 200k on the tablet. Jena and Sesame answer these queries with different delays. For example, Sesame answers query 5 faster than JenaTDB, but it responds more slowly in query 11. On the tablet, the performance of RDF-BDB is much slower than RDF-OTG and TDBoid due to the latency in Berkeley DB. For example, it can not answer query 1 and query 5

[5] https://beagleboard.org/black
[6] https://en.wikipedia.org/wiki/Nexus_7_(2012)

less than 20 seconds. Overall, in our test, 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.



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

Fig. 3. Query response time

We test the continuous query processor in the scenario of event processing [12]. It can process an average of 50 events per second on the Beagle Bone Black. The memory profiling shows that for the same size of data, our testing applications consume one-third memory as Jena does and a half as much as Sesame requires.

8 Reflections

This work aims to adapt embedded database management for efficient processing RDF data on embedded devices. This work is based on the understanding of database management techniques and embedded devices' architecture and RDF data processing. Our initial results promisingly show that database optimisations can support better scalability and performance for RDF data on embedded devices.

From the experiment results, we have found interesting opportunities to improve our system. For example, in the SPARQL query processor, the algorithm of Sesame can be applied to execute query 5, and in query 11 we use Jena's. Furthermore, a scalable reasoner can be achieved by leverage the high throughput storage to store the materialised triples which are derived from forwarding rules. It can be assumed that using a persistent triple storage to store derived triples will reduce the memory consumption and the cost of re-materialising data. If the RDF Dictionary on different devices can be efficiently synchronised, the transfer of RDF data between devices is fastened since only the encoded integer need to be sent.

Finally, we believe our software framework could support a scalable and efficient RDF data processing for pervasive and IOTs applications.

Acknowledgement: This thesis is funded by Irish Research Council under Grants No. GOIPG/2014/917 and supervised by Dr. Danh Le-Phuoc and Dr. Conor Hayes.

References

1. A. Ailamaki. Databases and Hardware: The Beginning and Sequel of a Beautiful Friendship. *Proceedings of the VLDB*, 2015.
2. 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.
3. P. Barnaghi, W. Wang, C. Henson, and K. Taylor. Semantics for the internet of things: Early progress and back to the future. *Int. J. Semant. Web Inf. Syst.*, 2012.
4. C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. *International Journal on Semantic Web and Information Systems*, 2001.
5. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame : A generic architecture for storing and querying rdf and rdf schema. *ISWC*, 2002.
6. M. D'Aquin, A. Nikolov, and E. Motta. How much semantic data on small devices? *EKAW*, 2010.
7. H. Hasemann, A. Kroller, and M. Pagel. RDF provisioning for the internet of things. *International Conference on the Internet of Things*, 2012.
8. H. Hasemann, A. Kroller, and M. Pagel. The Wiselib TupleStore : A Modular RDF Database for the Internet of Things. 2014.
9. Hector Garcia-Molina, J. D. Ullman, and J. Widom. *Database systems The Complete Book*. 1997.
10. I. Koltsidas and S. D. Viglas. Data management over flash memory. *SIGMOD*, 2011.
11. D. Le-phuoc. *A Native and Adaptive Approach for Linked Stream Data Processing*. PhD thesis, NUI Galway, 2013.
12. D. Le-Phuoc, M. Dao-Tran, A. Le-Tuan, M. Nguyen-Duc, and M. Hauswirth. Grand Challenge : RDF Stream Processing with CQELS Framework for Real-time Analysis. *DEBS*, 2015.
13. D. Le-phuoc, A. Le-tuan, G. Schiele, and M. Hauswirth. Querying Heterogeneous Personal Information on the Go. *ISWC*, 2014.
14. D. Le-phuoc, J. X. Parreira, and V. Reynolds. RDF On the Go: An RDF Storage and Query Processor for Mobile Devices. In *ISWC*, 2010.
15. D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac. Internet of things: Vision, applications and research challenges.
16. M. Miraoui., C. Tadj, and Chokri ben Armar. Architecuture survey of context-aware system in pervasive computing environment. *UbiComp*, 2008.
17. A. Nori. Mobile and embedded databases. In *SIGMOD*, 2007.
18. A. Owens. *Using Low Latency Storage to Improve RDF Store Performance*. PhD thesis, University of Southampton, 2011.
19. M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 2001.
20. B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. *WMCSA*, 1994.
21. M. Sharma. Compression Using Huffman Coding. *IJCSNS*, 2010.
22. T. Strang and C. Linnhoff-Popien. A Context Modeling Survey. *UbiComp*, 2004.
23. K. Wilkinson, C. Sayers, H. Kuno, and D. Reynolds. Efficient RDF storage and retrieval in Jena2. *SWDB*, 2003.