Towards Efficient Dissemination of Linked Data in the Internet of Things

Yongrui Qin*, Quan Z. Sheng*, Nickolas J.G. Falkner*, Ali Shemshadi*, Edward Curry†

*School of Computer Science
The University of Adelaide, Australia
{yongrui.qin,michael.sheng,nickolas.falkner,ali.shemshadi}
@adelaide.edu.au

†Insight
NUI Galway, Ireland
ed.curry@insight-centre.org

ABSTRACT

The Internet of Things (IoT) envisions smart objects collecting and sharing data at a global scale via the Internet. One challenging issue is how to disseminate data to relevant data consumers efficiently. In this paper, we leverage semantic technologies which can facilitate machine-to-machine communications, such as Linked Data, to build an efficient information dissemination system for semantic IoT. The system integrates Linked Data streams generated from various data collectors and disseminates matched data to relevant data consumers based on Basic Graph Patterns (BGPs) registered in the system by those consumers. To efficiently match BGPs against Linked Data streams, we introduce two types of matching, namely *semantic matching* and *pattern matching*, by considering whether the matching process supports semantic relatedness computation. Two new data structures, namely MVR-tree and TP-automata, are introduced to suit these types of matching respectively. Experiments show that an MVR-tree designed for semantic matching can achieve a twofold increase in throughput compared with the naive R-tree based method. TP-automata, as the first approach designed for pattern matching over Linked Data streams, also provides two to three orders of magnitude improvements on throughput compared with semantic matching approaches.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Selective dissemination of information*

Keywords

Linked data, information dissemination, query index

1. INTRODUCTION

In the era of the Internet of Things (IoT) [9], it is envisioned that smart objects collect and share data at a global scale via the Internet. In this context, semantic technologies such as Linked Data, which aim to facilitate machine-to-machine communications, play an increasingly important role [2]. Due to the large amount of data produced by various kinds of things, one challenging issue is how to efficiently disseminate data to relevant data consumers.

In this paper, we propose an efficient information dissemination system for semantic IoT by leveraging semantic technologies, such as Linked Data. Our system integrates Linked Data streams (in RDF format) generated from various data collectors. Data consumers can register their interest in the form of Basic Graph Patterns (BGPs) in the system. Based on these BGPs, the system disseminates matched Linked Data to relevant users. This work focuses on how to efficiently match a large number of BGPs against Linked Data streams.

Before introducing motivation, we identify three types of matching between Linked Data and BGPs as follows:

- *Match estimation* is typically used in source selection systems. It provides an estimation on how well a Linked Data source would match a given BGP. *Match estimation* may provide false negative and false positive match results. Recent work on data summaries on Linked Data [6] is an example of *match estimation*.
- *Semantic matching* aims to match semantically related RDF triples against BGPs. It may provide false positive match results but not false negative. Approximate event matching [7] applies *semantic matching*.
- *Pattern matching* means individual component matching between RDF triples and BGPs. It does not consider semantic relatedness between an RDF triple and a BGP. Similar to *semantic matching*, it may return false positive match results but not false negative. An example of *pattern matching* is recent work on stream reasoning [1].

In our Linked Data dissemination system, in order to disseminate high-quality information to data consumers (or subscribers), we only consider *semantic matching* and *pattern matching* as they will not return false negative match results. Or in other words, no matched triples will be missed.

Motivation. Recent work on data summaries on Linked Data [6] transforms RDF triples into numerical space. Then data summaries are built upon numerical data instead of strings as summarizing numbers is more efficient than summarizing strings. In order to transform triples into numbers,

CIKM'14, November 3–7, 2014, Shanghai, China.

http://dx.doi.org/10.1145/2661829.2661889.

hash functions are applied on the individual components `(s, p, o)` of triples. Thus a derived triple of numbers can be considered as a 3D point. In this way, a set of RDF triples can be mapped into a set of points in 3D space. To facilitate query processing over data summaries, a spatial index named QTree [6], which is evolved from standard R-tree [5], is adopted as the basic index. Data summaries are designed mainly for indexing various Linked Data sources and used for identifying relevant sources for a given query.

However, data summaries are not suitable for our Linked Data dissemination system. Firstly, data summaries techniques, such as QTree, do not consider variables in the BGPs but only RDF triples with concrete strings. QTree only indexes points in a 3D space while our system is required to index points, lines or even planes in a 3D space, depending on the number of variables in BGPs. Further, since data summaries are concise and imprecise representations of data sources [6], they just provide *match estimation*. Hence, query evaluation on them would return false negative results, which is not allowed in our system.

Moreover, existing work on *pattern matching*, such as stream reasoning [1], does not support large scale query evaluation but focuses on evaluation of a single query over the streaming Linked Data. Therefore, the issue of supporting *pattern matching* over a large number of BGPs against Linked Data streams remains open.

In this paper, we introduce two techniques, namely Multi-Version R-tree (MVR-tree) and Triple Pattern automata (TP-automata), to index a large collection of BGPs, which can support *semantic matching* and *pattern matching* respectively. Experiments show that MVR-tree can achieve a twofold increase in throughput for *semantic matching* compared with the naive R-tree based method. Moreover, TP-automata, as the first approach designed for *pattern matching* between a large number of queries and Linked Data streams, can achieve up to three orders of magnitude improvements on throughput compared with *semantic matching* approaches.

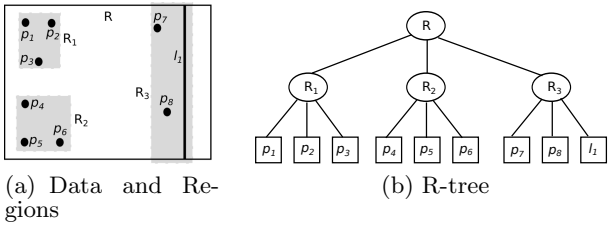
2. LINKED DATA DISSEMINATION SYSTEM

System Overview. In our Linked Data dissemination system, when the user queries (BGPs) are registered, all queries will be transformed into spatial objects in a 3D space. Then a suitable index will be constructed for efficient evaluation between Linked Data streams and user queries. Before matching starts, RDF triples in the data streams will be mapped into data points in the same 3D space first. Then, these data points will be matched with BGPs represented as spatial objects in the constructed indexes. Finally, matched triples will be forwarded to their subscribers.

User Queries. Basic Graph Patterns (BGPs) [6] are adopted as user queries in our system. The possible triple patterns in a BGP are: 1) `(#s, #p, #o)`, 2) `(?s, #p, #o)`, 3) `(#s, ?p, #o)`, 4) `(#s, #p, ?o)`, 5) `(?s, ?p, #o)`, 6) `(?s, #p, ?o)`, 7) `(#s, ?p, ?o)`, and 8) `(?s, ?p, ?o)`. Note that, `?` denotes a variable while `#` denotes a constant.

Similar to data summaries, we apply hash functions[1] to map these patterns into numerical values. These numerical values can be regarded as *coordinates* in a 3D space.

[1]There are many different hash functions. For more details, please refer to [6].



(a) Data and Regions (b) R-tree

Figure 1: Two-dimensional R-tree example

Specifically, given a hash function f, a triple pattern `(#s, #p, #o)` can be mapped into a 3D point $(f$`(#s)`$, f$`(#p)`$, f$`(#o)`$)$. We call such a point mapped from a triple pattern a *data point* in order to differ it from other points in the 3D space. Using this approach, a set of triples or triple patterns without variables can be mapped into a set of 3D data points. To map triples with variables, the hash function f works in the following way: f will map a variable to the whole range of a coordinate axis. For example, pattern 2 `(?s, #p, #o)` will be mapped into $(x,$ `f(#p)`$, f$`(#o)`$)$, which is a line in the 3D space. Triple patterns with more variables can be mapped in a similar way.

Clearly, pattern 1 can be mapped into a 3D data point. In order to match pattern 1 with a triple in the data stream, three coordinates of pattern 1's data point must match with those of the triple's data point altogether. Patterns 2 to 4 can be mapped into lines in the 3D space. To match a triple with patterns 2 to 4, only two coordinates have to be matched. Patterns 5 to 7 can be mapped into planes. Similarly, to match a triple with patterns 5 to 7, only one coordinate has to be matched. It should be noted that we do not consider to index pattern 8 in our work, as it will match all the triples in the Linked Data stream directly.

2.1 MVR-tree for Semantic Matching

Locality Sensitive Hashing (LSH) has been used to place similar documents into the same bucket of a hash table [8]. Similar technique can be used to hash RDF triples in our semantic matching.

Inspired by QTree [6], a naive solution to support semantic matching is to use R-Tree [5]. R-tree is a hierarchical structure and is used to index spatial objects. It consists of nodes representing regions in the data space. The region of a node always covers all the regions of its child nodes. The region of a node is also a Minimum Bounding Box (MBB). Leaf nodes store actual spatial objects. An 2D R-tree example is depicted in Figure 1. There are seven points and one line in Figure 1(a). These spatial objects are bounded by three regions, namely R_1, R_2 and R_3. These three regions are further bounded by region R. Figure 1(b) shows the R-tree structure for spatial objects in Figure 1(a). R-tree can support Nearest Neighbor (NN) searches efficiently [5] which in turn can support *semantic matching* efficiently after mapping BGPs into spatial objects in a 3D space.

Figure 1(a) shows that the region which contains a line usually occupies large space, such as region R_3 in the figure. Since regions that contain lines will occupy large space, if there are many lines in the space, the regions bounding these lines would overlap each other with high probability. If there are many overlapped regions in an R-tree, MBBs would overlap with each other with high probability and

the R-tree performance would degrade greatly [5]. What is worse, there may be planes to bound, which further magnifies this problem. As a result, if we use R-tree to directly index spatial objects mapped from a set of BGPs with variables, the performance of R-tree would become an issue.

Based on this observation, we introduce Multi-Version R-tree (MVR-tree) to alleviate performance deterioration on R-tree. MVR-tree is an R-tree variant. As mentioned, BGPs can be mapped as points, lines, or planes in a 3D space, depending on the number variables they have. In MVR-tree, for BGPs mapped as points, we still use a 3D R-tree to index them because all three coordinates of them require to be matched. For BGPs mapped as lines, we use 2D R-trees to index them because only two coordinates require to be matched. Since there are three types of lines (in parallel with x coordinate, with y coordinate, or with z coordinate in the 3D space), we need to use three 2D-trees to index these three types of lines. For BGPs mapped as planes, we use one-dimensional R-trees to index them, which are B-trees, by definition [3]. Similar to indexing lines, we also need to use three B-trees to index the three types of planes.

To be more specific, in a MVR-tree, a 3D R-tree is used to index BGPs without variables (pattern `(#s, #p, #o)`), three 2D R-trees are used to index BGPs with only one variable (pattern `(?s, #p, #o)`, `(#s, ?p, #o)`, and `(#s, #p, ?o)`), and three B-trees are used to index BGPs with two variables (pattern `(?s, ?p, #o)`, `(?s, #p, ?o)`, and `(#s, ?p, ?o)`). We call describe these trees as versioned trees, which together form an MVR-tree index.

To evaluate an RDF triple with an MVR-tree, the system will need to check whether the data point mapped from the triple matches any points in at least one versioned tree of the MVR-tree. Since there only points in each versioned tree of the MVR-tree, the match process should be quite efficient.

Since R-tree can support NN queries efficiently, we can adapt its variant MVR-tree to support semantic matching where the most similar triples should be returned. Taking advantage of the LSH techniques, we can return NN query results as the most similar triples for a BGP query.

2.2 TP-automata for Pattern Matching

Automata techniques have been adopted to process XML data streams [4] They are mainly based on languages with SQL-like syntaxes, and relational database execution models adapted to process streaming data. In our system, to support *pattern matching*, we also apply automata to match each individual component of a triple with its counterparts of a BGP efficiently, which we call Triple Pattern automata (TP-automata).

Firstly, as mentioned, operating on numbers is more efficient than operating on strings. Similar to MVR-tree, we also map BGPs into spatial objects in 3D space. However, the difference is that we treat variables in a BGP as a universal match indicator, e.g. represented by "?". This indicator will be mapped into a fixed and unique numerical value but not the whole range of a specific coordinate axis. Such unique numerical values will be treated differently as well later in the triple evaluation process.

Figure 2 depicts the construction process of TP-automata. Firstly, user queries will be transformed into triple pattern state machines as shown in the middle of Figure 2. As can be seen from the figure, each triple state machine contains an initial state, two internal states, one final state and three

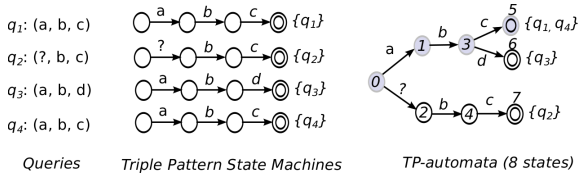


Figure 2: TP-automata

transitions. In the figure, the first circle of a state machine represents the initial state, the next two circles represent the two internal states and the doubled circle represents the final state. The three arrows associated with conditions are three transitions between different states. Similar to [4], these state machines can be combined into one machine by exploiting shared common states with same transitions. The combined machine, TP-automata, is shown on the right of Figure 2. The shaded circles represent combined states.

To perform *pattern matching* over TP-automata, triples in the Linked Data stream will be firstly mapped into 3D points. For example, suppose a triple `(s, p, o)` is mapped into a 3D point `(a, b, c)`. The system will match it against TP-automata in the following process. It firstly checks the initial state of TP-automata and looks for state transitions with condition `a` or condition `?`. Following the state transitions, state 1 and state 2 become the current active states at the same time. It then looks for state transitions with condition `b` or `?` from state 1 and state 2. Following the transitions, state 3 and state 4 become active states. Finally, following transitions with condition `c` or `?` from state 3 and state 4, two final states, state 5 and state 7, are reached. By checking both final states, the system returns $\{q_1, q_2, q_4\}$ as the matching results. The match process stops if and only all current active states are final states or states with no satisfied transition.

3. EXPERIMENTS

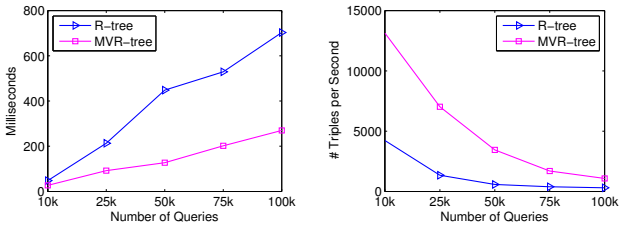
3.1 Experimental Setup

The data set used in our experiment is a subset of the current version of the English DBpedia[2]. It contains resources of type `dbpedia-owl:Event`. Each event is a triple in the form `<eventURI, rdf:type, dbpedia-owl:Event>`. An example of an event URI is <`http://dbpedia.org/resource/Battle_of_Brentford_(1642)`>. There are approximately 400,100 triples in the dataset.

As an initial work, we used simple BGPs as queries in the experiment and we leave extending our system to support complex BGPs or join queries as our future work. We randomly generated BGPs using the seven patterns mentioned in Section 2 based on our dataset.

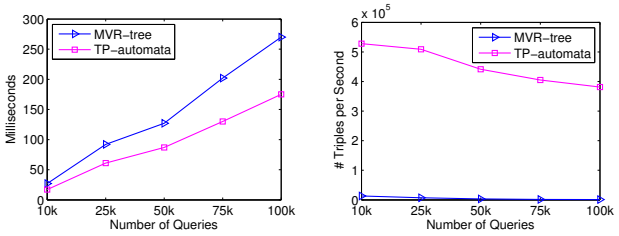
We evaluated the performance of our methods in terms of `Average Construction Time` (in milliseconds) of the indexes and `Average Throughput` (in number of triples per second). We performed experiments in two scenarios. One was for *semantic matching* by comparing MVR-tree and R-tree methods, and the other was for *pattern matching* by comparing TP-automata and MVR-tree methods because TP-automata is the first method to evaluate Linked Data streams over a large number of user queries simultaneously.

[2] `http://downloads.dbpedia.org/3.8/en/`



(a) Average Construction Time (b) Average Throughput

Figure 3: Performance on Semantic Matching



(a) Average Construction Time (b) Average Throughput

Figure 4: Performance on Pattern Matching

3.2 Experimental Results

3.2.1 Semantic Matching

Figure 3 presents performance on *semantic matching* by comparing R-tree and MVR-tree. Average construction time is depicted in Figure 3(a). We can see that construction time of both indexes increases linearly as the number of queries grows. Constructing MVR-tree indexes takes less time than R-tree. The reason for this is that there are multiple versioned trees in an MVR-tree index and each versioned tree is smaller than R-tree. When a new BGP is inserted into MVR-tree, only one versioned tree will be updated. Since versioned trees are smaller compared with R-tree, the update process in MVR-tree is more efficient.

Figure 3(b) shows the throughput results of R-tree and MVR-tree. It is evident that MVR-tree at least three times of R-tree's throughput under all query number settings. The root cause is that BGPs with variables are large spatial objects in R-tree leading to deteriorative throughput performance on R-tree. On the other hand, as described in Section 2, MVR-tree only has points as spatial objects in all its versioned trees. Hence NN searches on each versioned tree are quite efficient.

3.2.2 Pattern Matching

The performance of *pattern matching* on MVR-tree and TP-automata is presented in Figure 4. Firstly, average construction time is compared in Figure 4(a). TP-automata can be constructed faster than MVR-tree because TP-automata does not need to split nodes while MVR-tree would need to split some big nodes and adjust node levels and relationships during the construction process.

Throughput performance of *pattern matching* is presented in Figure 4(b). It shows very large differences between MVR-tree and TP-automata. TP-automata shows two to three orders of magnitude better throughput performance compared with MVR-tree. The root cause is that MVR-tree is designed for *semantic matching* in a hashing space (e.g., handling NN queries) while TP-automata is designed for *pattern matching*. To perform a *pattern matching* on MVR-tree is actually equivalent to processing an NN query while TP-automata just performs $O(1)$ lookups at each state transition. Hence, the TP-automata approach shows superior performance over the MVR-tree approach.

4. CONCLUSIONS

In this paper, we have leveraged semantic technologies, such as Linked Data, to build an efficient information dissemination system for semantic IoT. We have identified three types of match, which are *match estimation*, *semantic matching*, and *pattern matching*, to establish the basic requirements for building a Linked Data dissemination system. In order to efficiently match a large number of BGPs against Linked Data streams, we have proposed two index schemes, namely MVR-tree and TP-automata. MVR-tree is a Multi-Version R-tree designed for efficiently processing *semantic matching* while TP-automata is an automata based method designed for efficient *pattern matching*. Experiments show that our methods outperform existing indexing methods that are derived from R-tree. Specifically, in *semantic matching*, MVR-tree achieves a twofold increase in throughput compared with the traditional R-tree. In *pattern matching* the TP-automata approach outperforms the MVR-tree approach by up to three orders of magnitude in terms of throughput.

5. REFERENCES

[1] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic. EP-SPARQL: A Unified Language for Event Processing and Stream Reasoning. In *WWW*, pages 635–644, 2011.

[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] D. Comer. The ubiquitous b-tree. *ACM Comput. Surv.*, 11(2):121–137, 1979.

[4] Y. Diao, M. Altinel, M. J. Franklin, H. Zhang, and P. M. Fischer. Path Sharing and Predicate Evaluation for High-Performance XML Filtering. *ACM Trans. Database Syst.*, 28(4):467–516, 2003.

[5] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*, pages 47–57, 1984.

[6] A. Harth, K. Hose, M. Karnstedt, A. Polleres, K.-U. Sattler, and J. Umbrich. Data Summaries for On-Demand Queries over Linked Data. In *WWW*, pages 411–420, 2010.

[7] S. Hasan and E. Curry. Approximate Semantic Matching of Events for the Internet of Things. *ACM Trans. Internet Techn.*, 14(1):2, 2014.

[8] S. Petrovic, M. Osborne, and V. Lavrenko. Streaming First Story Detection with application to Twitter. In *HLT-NAACL*, pages 181–189, 2010.

[9] Y. Qin, Q. Z. Sheng, N. J. G. Falkner, S. Dustdar, H. Wang, and A. V. Vasilakos. When Things Matter: A Data-Centric View of the Internet of Things. *CoRR*, abs/1407.2704, 2014.