Batch Matching of Conjunctive Triple Patterns over Linked Data Streams 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 consumers efficiently. This paper leverages semantic technologies, such as Linked Data, which can facilitate machine-to-machine (M2M) communications 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 conjunctive triple pattern queries registered in the system by the consumers. We also design a new data structure, *CTP-automata*, to meet the high performance needs of Linked Data dissemination. We evaluate our system using a real-world dataset generated from a Smart Building Project. With CTP-automata, the proposed system can disseminate Linked Data at a speed of an order of magnitude faster than the existing approach with thousands of registered conjunctive queries.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information filtering*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Selective dissemination of information*

Keywords

Linked data, information dissemination, query index

1. INTRODUCTION

As of 2012, 2.5 quintillion (2.5×10^{18}) bytes of data were being created daily[1]. In the Internet of Things, connecting all of the things that people care about in the world becomes possible [12]. However, all these things will produce much more data than nowadays. The volumes of data are vast, the generation speed of data is fast and the data/information space is global. Indeed, IoT is one of the major driving forces for *big data analytics*. Given the scale of IoT, topics such as distributed processing, real-time data stream analytics, and event processing are all critical, and need to be revisited in order to improve upon existing technologies for applications of this scale [8, 2]. In this context, semantic technologies such as Linked Data (see http://linkeddata.org/), which aim to facilitate machine-to-machine (M2M) communications, play an increasingly important role [3]. Linked Data is part of a growing trend towards highly distributed systems, with thousands or potentially millions of independent sources providing structured data. 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.

To deal with such challenge, it is imperative to efficiently retrieve the most relevant data from the big data generated in IoT and effectively extract useful information (e.g., in the process converting "data" into "information" or "knowledge"). We propose in this paper an efficient data dissemination system for semantic IoT by leveraging semantic technologies, such as Linked Data. Our system will be very helpful and efficient in the retrieval of relevant data from the deluge of IoT data, which can then facilitate the extraction of required information. The system firstly integrates data generated from various data collectors. Then it transforms all the data into Linked Data streams in Resource Description Framework (RDF) format (see www.w3.org/RDF). Meanwhile, data consumers can register their interests in the form of *conjunctive triple pattern queries* [15, 9] composed of triple patterns in the system. Based on these conjunctive queries, the system disseminates matched Linked Data to relevant users. After receiving relevant data, these users can further make use of the data to extract information for their own purposes, such as environment monitoring, event detection, complex event processing, and so on. It should be noted that we will not discuss the data processing at the user side, instead we focus ourselves on how to efficiently match a large number of conjunctive queries against Linked Data streams in batch mode. We highlight our main contributions in the following.

- We identify new Linked Data dissemination needs in the context of the Internet of Things, which requires to process continuous data requests in batch mode efficiently.

[1]http://www-01.ibm.com/software/data/bigdata/

SSDBM'15, June 29 – July 01, 2015, La Jolla, CA, USA
 ISBN 978-1-4503-3709-0/15/06 ...$15.00.
DOI: http://dx.doi.org/10.1145/2791347.2791364.

- We develop a novel data structure, Conjunctive Triple Pattern automata (CTP-automata), for efficiently matching Linked Streams against a large number of conjunctive triple pattern queries based on automata techniques. We also develop novel techniques to evaluate conjunctive queries efficiently.

- We conduct extensive experiments using a real-world dataset generated in a Smart Building Project. The results show that our proposed system can disseminate Linked Data at a speed of an order of magnitude faster than the existing approaches with thousands registered conjunctive queries.

2. LINKED DATA DISSEMINATION SYSTEM

2.1 System Overview

Figure 1 shows an overview of our system in the smart city scenario. We assume that data generated by all kinds of things will be represented in the form of Linked Data streams using RDF (for the purpose of facilitating machine-to-machine (M2M) communications). Our system consists of two main components: the *matching component* and the *index construction component*. Data consumers (humans and/or smart things in the city) can register their interests as user queries in the system. The index construction component constructs an index for all user queries. The matching component evaluates the incoming Linked Data streams against the constructed index for efficiently matching triples to the user queries. Finally, the system disseminates the matched data to relevant data consumers for their further processing.

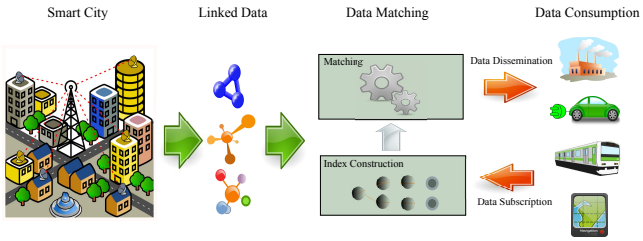

Figure 1: System Overview

User Queries. Similar to [15, 9], triple patterns are adopted as the basic units of user queries in our system. A triple pattern is an expression of the form `(s, p, o)` where `s` and `p` are URIs or variables, and `o` is a URI, a literal or a variable. The eight possible triple patterns 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)`. Here, `?` denotes a variable while `#` denotes a constant. Similar to data summaries in [7], we can also apply hash functions[2] to map these patterns into numerical values.

A user query can be expressed as a *conjunctive* triple pattern query composed of multiple triple patterns. Conjunctive queries can express data needs much more accurately compared to single triple pattern queries. A conjunctive query q has the form of:

$$?x_1, \ldots, ?x_n : (s_1, p_1, o_1)(s_2, p_2, o_2) \cdots (s_n, p_n, o_n)$$

where $?x_1, \ldots, ?x_n$ are variables, each (s_i, p_i, o_i) is a triple pattern, and each variable $?x_i$ appears in at least one triple pattern (s_i, p_i, o_i). Variables will always start with the '?' character. Variables $?x_1, \ldots, ?x_n$ are also called *answer variables* in order to distinguish them from other variables in the query.

Representations of Queries and Triples. In our Linked Data dissemination system, when the user queries (in the form of conjunctive triple pattern queries) are registered, all queries are transformed into numerical values. The reason for this is that the comparisons between numbers are faster than strings [7]. Note that, in such case, we will have three numbers for the three components in a query as described above. Then a suitable index can be constructed for efficient evaluation between Linked Data streams and user queries. Before a matching process starts, RDF triples in the data streams will be mapped into numerical values as well. Then, these numerical represented triples will be matched with conjunctive queries represented as numerical values in the constructed indexes.

2.2 CTP-automata for Conjunctive Query Matching

Automata techniques have been adopted to process XML data streams [5]. They are mainly based on languages with SQL-like syntax, 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 triple patterns in a conjunctive query efficiently. We call this approach *Conjunctive Triple Pattern automata* (CTP-automata).

As mentioned, operating on numbers is more efficient than operating on strings. Note that when we map triple patterns into numerical values, we treat variables in a triple pattern 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. This unique numerical value will be treated differently as well later in the triple evaluation process.

Construction of CTP-automata. Figure 2 depicts the construction process of CTP-automata. There are two conjunctive queries, $q_1 : (?x1, b, c)(?x1, d, e)$ and $q_2 : (?x2, b, c)$ $(?x2, d, e)(a, d, ?x2)$. Accordingly, there are two triple patterns in q_1 and three triple patterns in q_2. Firstly, all the triple patterns in the conjunctive queries will be transformed into triple pattern state machines as shown in the middle of Figure 2. As can be seen from the middle part of the figure, each triple state machine contains an initial state, two internal states, one final state, and three 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 represent three transitions between different states.

It is worth mentioning that we ignore variable names at this stage due to the fact that when processing triples in the Linked Data stream, at the first step, we have to evaluate

[2]There are many different hash functions that are suitable for this purpose. For more details, please refer to [7].

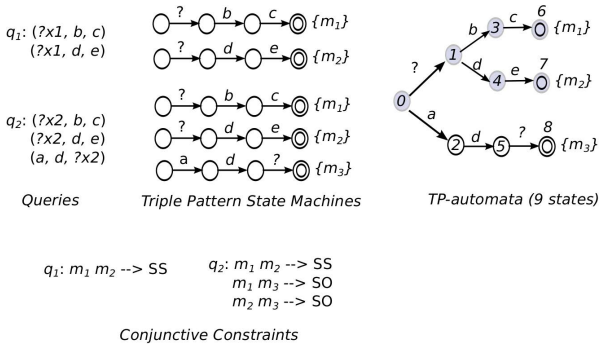

Figure 2: Index Structure and Conjunctive Constraints of CTP-automata

these triples one by one and that variable naming does not have any relationships between different conjunctive queries. For example, $(?x1, b, c)(?x1, d, e)$ and $(?x2, b, c)(?x2, d, e)$ actually refer to the same conjunctive query. However, the variable naming does matter within the same conjunctive query. For example, in $(?x1, b, ?x2)(?x2, d, e)$, variables $?x1$ and $?x2$ refer to different triple components. We leave the resolution of different variable names within the same conjunctive query in the later stage, called `Conjunctive Constraints Resolution` (CCR) stage. Before that, we need to evaluate each triple against each single triple state machine first, which is the `Triple Pattern Matching` (TPM) stage.

Similar to [5], the multiple single triple state machines shown in Figure 2 can be combined into one triple state machine by exploiting shared common states with same transitions. The combined machine, CTP-automata, is shown on the right of Figure 2. The shaded circles represent combined states. We can see from the figure that, although we have five single triple state machines, after the combination, the number of single triple state machines drops to three, which have been labeled as m_1, m_2 and m_3, respectively.

Matching Triple Streams against CTP-automata. During the TPM stage, in order to perform *pattern matching* over CTP-automata, when a triple `(a, b, c)` arrives, our system firstly checks the initial state of CTP-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, only state 3 becomes active state and there is no transition triggered from state 2. Finally, following the transition with condition `c` or `?` from state 3, one final state, state 6, is reached. By checking this final state, the system returns $\{m_1\}$ as the matching result. The matching process stops if and only if all current active states are final states or states with no satisfied transition.

At this TPM stage, the matching results are only intermediate results and the matched triples are just possible candidates which may satisfy some conjunctive queries. In order to determine which conjunctive queries have been satisfied, we need to further evaluate some `conjunctive constraints`, which will be detailed next.

Conjunctive Constraints Resolution (CCR) of CTP-automata. It should be noted that in order to match q_1 and q_2 in Figure 2, all triple patterns they contain must be matched first. Take query $q_1 : (?x1, b, c)(?x1, d, e)$ as an example. Suppose that triples t_1 and t_2 match triple patterns $(?x1, b, c)$ and $(?x1, d, e)$, respectively. To ensure that query q_1 can be satisfied by t_1 and t_2, we need to check first that whether we have $t_1.s = t_2.s$. We call such conditions as *conjunctive constraints* of a conjunctive query. All conjunctive constraints must be satisfied before we can assure that a conjunctive query is satisfied. As mentioned before, the conjunctive constraints checking occurs at the CCR stage.

In this paper, we have identified ten conjunctive constraints, including SS, PP, OO, SO, OS, SSPP, SSOO, PPOO, SOPP, OSPP. Constraint SS means that the subjects of two candidate triples must be matched. More details are shown in Table 1. These constraints can be used to determine whether a conjunctive query has been satisfied or not so far in the stream.

For example, in Figure 2, query q_1's conjunctive constraint is $m_1 m_2 - - >$SS and query q_2's conjunctive constraints are $m_1 m_2 - - >$SS, $m_1 m_3 - - >$SO and $m_2 m_3 - - >$SO. Suppose that triples t_1, t_2, t_3 match triple pattern machines m_1, m_2, m_3, respectively. According to Table 1, for t_1, t_2 to satisfy q_1, we need to have $t_1.s = t_2.s$. Similarly, for t_1, t_2, t_3 to satisfy q_2, we need to have $t_1.s = t_2.s$, $t_1.s = t_3.o$ and $t_2.s = t_3.o$.

Dynamic Maintenance of the Matching Candidate List. In order to check conjunctive constraints, triples in the stream that match some triple pattern machines will be buffered for this purpose. Since the Linked Stream can be considered infinite, the buffered triple lists for triple pattern machines may grow all the time. To avoid this issue, we need to specify a sliding window to confine our matching scope. That is, we only consider matching within the sliding window.

Figure 3 shows two sliding windows with size T: w_1 and w_2, where only the most recent T triples will be considered for our matching. In order to evaluate conjunctive constraints, we need to update the buffered candidate triple list each time a triple arrives in or leaves the window. In Figure 3, for w_1, we have got matching results for all three single triple pattern machines, $m1, m2, m3$, in Figure 2. After receiving a new tripe, t_{i+T}, the oldest triple t_i will be removed from all candidate lists that contain t_i. In this example, only candidate list for $m1$ contains t_i and hence t_i will be removed from that candidate list. Further, suppose the new arriving triple t_{i+T} will be matched with machine $m3$. Then t_{i+T} will be added to the candidate list for $m3$. It is obvious that, each time when the sliding window moves forward by one triple, we should consider all the buffered lists affected by the leaving triple and the joining triple in the sliding window to verify conjunctive constraints.

3. EXPERIMENTAL EVALUATION

In this section, we report our experimental evaluation of the proposed approach. We will first describe the experimental settings, and then report the experimental results.

3.1 Experimental Setup

The dataset used in our experiments was generated in a Smart Building Energy Project [4]. The energy readings were collected from 4–19 August 2014. In total, there are

Table 1: Conjunctive Constraints

Conjunctive Constraints	Description	Checking Details
SS	The subjects of two candidate triples must be matched	$t_1.s = t_2.s$
PP	The predicates of two candidate triples must be matched	$t_1.p = t_2.p$
OO	The objects of two candidate triples must be matched	$t_1.o = t_2.o$
SO	The subject of a candidate triple in the first pattern machine matches the object of a candidate triple in the second pattern machine	$t_1.s = t_2.o$
OS	The object of a candidate triple in the first pattern machine matches the subject of a candidate triple in the second pattern machine	$t_1.o = t_2.s$
SSPP	The conjunction of both SS and PP constraints	$t_1.s = t_2.s$ and $t_1.p = t_2.p$
SSOO	The conjunction of both SS and OO constraints	$t_1.s = t_2.s$ and $t_1.o = t_2.o$
PPOO	The conjunction of both PP and OO constraints	$t_1.p = t_2.p$ and $t_1.o = t_2.o$
SOPP	The conjunction of both SO and PP constraints	$t_1.s = t_2.o$ and $t_1.p = t_2.p$
OSPP	The conjunction of both OS and PP constraints	$t_1.o = t_2.s$ and $t_1.p = t_2.p$

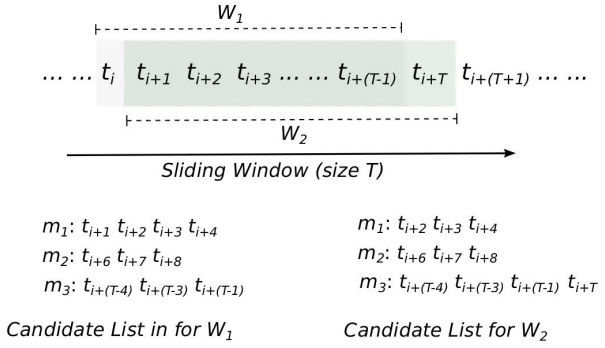

Figure 3: Maintenance of Candidate Triple List

Parameter	Range	Default	Description
Query Number	100 to 2000	1000	The number of conjunctive triple pattern queries
Pattern Number	1 to 5	3	The maximum number of triple patterns in a query
Window Size	10 to 500	100	The window size, in terms of number of triples, for the evaluation of conjunctive triple pattern queries

Table 2: Workload Parameters for the Experiments

around 6.2 million triples in the dataset. A more detailed description of the schema for the data and the energy readings can be found in [11].

We used random walk method to generate conjunctive triple pattern queries in the experiments according to the data graph of the event data. The details of parameters we used for generating these queries are shown in Table 2. The parameters include `query number`, `pattern number`, and `window size`.

We evaluated the performance of our approach in terms of *average construction time* (in milliseconds) of the indexes and *average throughput* (in number of triples per second). We compared hash-based implementation (i.e., mapping triples and queries into numerical values, denoted as *HashMat* in the figures) with string-based implementation (i.e., using triples and queries as it is, denoted as *StringMat* in the figures). We also compared our methods with an existing approach, CQELS [10], which supports parallel query evaluation on Linked Data streams. Both CTP-automata engines and CQELS[3] were all implemented on Java Platform Standard Edition 7 running on Linux (Ubuntu 12.10, 64-bit Operating System), with quad-core CPU@2.20GHz and 4 GB main memory. We ran each experiment 10 times in order to ensure consistency of results and reported the average experimental results.

[3]The source code and documentation of CQELS can be obtained via `http://code.google.com/p/cqels/`

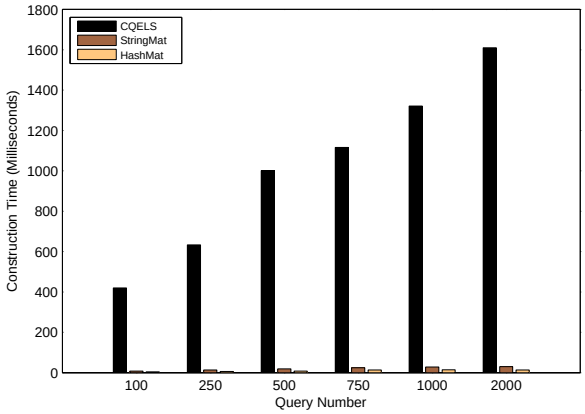

Figure 4: Average Construction Time

3.2 Performance Study

Construction Time. The average construction times of CTP-automata engines and CQELS engine is presented in Figure 4. The construction times for both hash-based CTP-automata matching engine (HashMat) and string-based CTP-automata matching engine (StringMat) are close to each other in most settings and are always under 50 milliseconds.

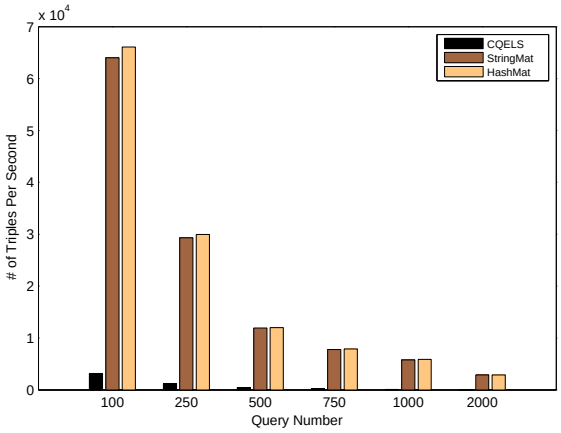

Figure 5: Average Throughput Evaluation by Varying Query Number

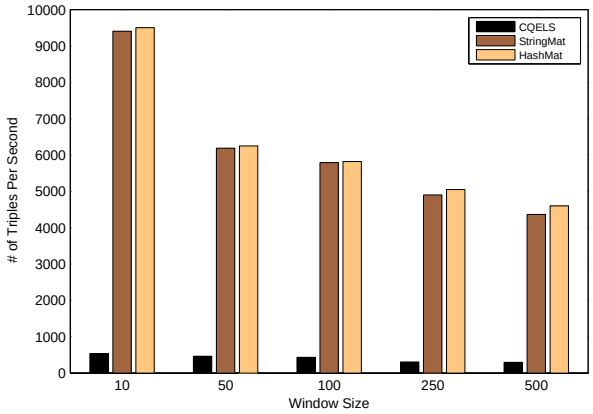

Figure 7: Average Throughput Evaluation by Varying Window Size

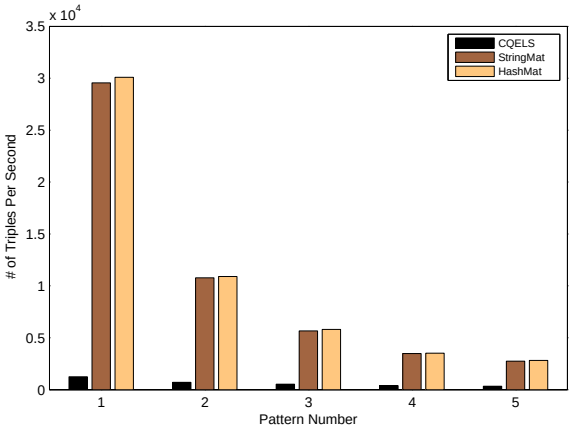

Figure 6: Average Throughput Evaluation by Varying Pattern Number

The construction of the string-based indexes takes slightly longer time. By contrast, the construction times of CQELS are much longer than CTP-automata engines. The main reason is that CQELS has to parse the conjunctive triple pattern queries using a SPARQL-like parser and then register the parsed queries in the processing engine. As shown from Figure 4, the construction times of CQELS grow linearly with the number of conjunctive queries. When the query number is 100, the construction time is around 400 milliseconds. When the number of queries increases to 2000, the construction time reaches above 1610 milliseconds. This indicates that the construction of our CTP-automata engines is very fast.

Throughput. The throughput performance of *pattern matching* under varying query numbers is depicted in Figure 5. It shows some similarities between HashMat and StringMat. In most cases, HashMat shows slightly better throughput speed compared with StringMat. This indicates that although comparisons on strings are slower than those on numbers, the differences betwen HashMat and StringMat are negligible. The main reason for this is that the evaluation process of conjunctive queries spends a large proportion of time to evaluate the conjunctive constraints on each query and both HashMat and StringMat use the same strategy to evaluate all these conjunctive constraints.

However, when compared with CQELS, both HashMat and StringMat outperform CQELS significantly. To be specific, when the number of conjunctive queries is 100, the throughput of HashMat and StringMat is more than 64,000 triples per second, and for CQELS, just slightly more than 3,000. When the number of conjunctive queries is 2,000, the throughput of HashMat and StringMat drops to slightly below 3,000 triples per second while CQELS has a throughput about just 50 triples per second. From Figure 5, we can observe that (1) HashMat and StringMat are normally 20 to 50 times faster than CQELS; (2) the throughput of HashMat, StringMat and CQELS all drops greatly when increasing the number of conjunctive queries. This also indicates that the evaluation of conjunctive constraints on each query takes a large amount of time and is difficult to share evaluation results between different conjunctive queries.

Figure 6 further demonstrates this finding. In the figure, we vary the maximum number of patterns of each conjunctive query. For the same amount of conjunctive queries, when the pattern number is only 1, the throughput of HashMat and StringMat is around 30,000 triples per second and for CQELS, it is around 1,200 triples per second, which is more than 20 times slower. When the pattern number is set to 5, the throughput of HashMat and StringMat drops to slightly lower than 3,000 triples per second and for CQELS, it drops to around 300 triples per second. This confirms that the evaluation of conjunctive constraints is time consuming. Similarly, HashMat and StringMat are both around an order of magnitude faster than CQELS.

Finally, Figure 7 depicts the effect of window size, which is varied from 10 to 500. From the figure, we can observe that when the window size increases from 10 to 50, the throughput of HashMat and StringMat drops from 9,500 triples per second to around 6,200 triples per second. But when the window size increases from 50 to 500, the throughput of HashMat and StringMat only drops to around 4,500 triples per second. This indicates that the window size does not

affect the throughput heavily like query number and pattern number. Similar effect of window size can be found on CQELS. When the window size increases from 10 to 500, the throughput of CQELS drops from around 500 triples per second to slightly lower than 300 triples per second. Still, HashMat and StringMat are both an order of magnitude faster than CQELS.

From our experimental study, we can conclude that CTP-automata indexes for conjunctive queries can be constructed much faster than the query registration process in CQELS. More importantly, CTP-automata (HashMat and StringMat) significantly outperforms CQELS in terms of throughput. Further, by using hashing techniques, HashMat performs slightly better than StringMat.

4. RELATED WORK

In terms of triple pattern matching, a large body of work which focuses on optimizing individual query processing has also been put forward [9, 6, 16, 14]. Specifically, the problem of evaluating conjunctive triple pattern queries is studied in [9] in the context of Peer-to-Peer (P2P) networks. In [6], an indexing technique for efficient join processing on RDF graphs is proposed. The index is constructed upon RDF data directly but not join queries. Similarly, the work in [16] focuses on optimizing the processing of conjunctive triple pattern queries, especially star-shaped group based queries individually. Furthermore, optimization on RDF graph pattern matching on MapReduce is also studied in [14]. However, the common problem shared by these research efforts is that, they have not considered the scenarios of optimizing conjunctive triple pattern queries in batch mode, which is the focus of our work in this paper.

Some existing work on *pattern matching* of Linked Data, such as stream reasoning [1] and Linked Data stream processing [10], does not support large-scale query evaluation but focuses on the evaluation of a single query or a small number of parallel queries over the streaming Linked Data. Other existing work only studies pattern matching of multiple single triple patterns [13, 11], but not multiple conjunctive triple patterns. Therefore, the issue of supporting *pattern matching* over a large number of conjunctive triple patterns against Linked Data streams still remains open in these approaches.

5. CONCLUSION

In this paper, we have leveraged semantic technologies, such as Linked Data, to build an efficient information dissemination system for semantic IoT. In order to efficiently match a large number of conjunctive triple pattern queries against Linked Data streams in batch mode, we have proposed CTP-automata, an automata-based method designed for efficient *pattern matching*. In our evaluation, we show that CTP-automata can disseminate Linked Data an order of magnitude faster than the existing approaches. This confirms the efficiency and effectiveness of our proposed approach. Our future work aims to support efficient matching for larger scales of conjunctive triple pattern queries, which would be a critical issue in the emerging Internet of Things.

6. 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, A. P. Sheth, and C. A. Henson. From Data to Actionable Knowledge: Big Data Challenges in the Web of Things. *IEEE Intelligent Systems*, 28(6):6–11, 2013.

[3] 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.

[4] E. Curry, S. Hasan, and S. O'Riain. Enterprise energy management using a linked dataspace for Energy Intelligence. In *SustainIT*, pages 1–6, 2012.

[5] 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.

[6] G. H. L. Fletcher and P. W. Beck. Scalable indexing of RDF graphs for efficient join processing. In *CIKM*, pages 1513–1516, 2009.

[7] 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.

[8] A. E. James, J. Cooper, K. G. Jeffery, and G. Saake. Research Directions in Database Architectures for the Internet of Things: A Communication of the First International Workshop on Database Architectures for the Internet of Things (DAIT 2009). In *BNCOD*, pages 225–233, Birmingham, UK, 2009. Springer.

[9] E. Liarou, S. Idreos, and M. Koubarakis. Evaluating Conjunctive Triple Pattern Queries over Large Structured Overlay Networks. In *ISWC*, pages 399–413, 2006.

[10] D. L. Phuoc, M. Dao-Tran, J. X. Parreira, and M. Hauswirth. A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data. In *ISWC*, pages 370–388, 2011.

[11] Y. Qin, Q. Z. Sheng, and E. Curry. Matching Over Linked Data Streams in the Internet of Things. *IEEE Internet Computing*, 19(3):21–27, 2015.

[12] 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.

[13] Y. Qin, Q. Z. Sheng, N. J. G. Falkner, A. Shemshadi, and E. Curry. Towards Efficient Dissemination of Linked Data in the Internet of Things. In *CIKM*, pages 1779–1782, 2014.

[14] P. Ravindra, H. Kim, and K. Anyanwu. An Intermediate Algebra for Optimizing RDF Graph Pattern Matching on MapReduce. In *ESWC, Part II*, pages 46–61, 2011.

[15] A. Seaborne. Rdql - a query language for RDF. In *W3C Member Submission*, 2001.

[16] M. Vidal, E. Ruckhaus, T. Lampo, A. Martínez, J. Sierra, and A. Polleres. Efficiently Joining Group Patterns in SPARQL Queries. In *ESWC, Part I*, pages 228–242, 2010.