Query for Streaming Information: Dynamic Processing and Adaptive Incremental Maintenance of RDF Stream

Xuanxing Yang*
School of Computer Science and Technology, Tianjin University
Tianjin, China
tdyxx@tju.edu.cn

ABSTRACT

Recently with dynamic information being ubiquitous on the Web, there have been efforts to extend RDF and SPARQL for representing streaming information and continuous querying functionalities, respectively. While existing works focusing on formalization and implementation of continuous querying process over RDF streams, little attention has deserved the problem of querying the complex temporal correlations among RDF stream tuples and the effective, scalable implementation of RDF stream processing system. To fill this gap, in this paper we propose CT-SPARQL and AIMRS, a language specifying for the compositional stream patterns and an architecture for adaptive incremental maintenance of RDF stream tuples defined in CT-SPARQL. We believe that this work will benefit a wide range of real-time analyzing and future predicting applications.

CCS CONCEPTS

• **Information systems** → **Resource Description Framework (RDF)**;

KEYWORDS

RDF Stream; Continuous Query; Stream Processing; Adaptive Incremental Maintenance

ACM Reference Format:
Xuanxing Yang. 2018. Query for Streaming Information: Dynamic Processing and Adaptive Incremental Maintenance of RDF Stream. In *WWW '18 Companion: The 2018 Web Conference Companion, April 23–27, 2018, Lyon, France*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3184558.3186573

1 PROBLEM

Recently with the development of a large amount of real-time applications, a new kind of streaming information such as GPS signal, real-time sensor readings and social media comments, has merged on the Web. Different from other persistent relations among resources on the Web, this kind of information are temporal statements for real-time conditions of various aspects such as traffic, sensor network, consensus, and are continuously delivered to processing systems in the form of data streams. Once get timely processed and appropriately explained, the temporal information carried by these heterogeneous data streams, will motivate a wide range of real-time analyzing and future predicting applications. However, the dynamic nature of streaming information has posed several challenges for its applications.

On the semantic level, to exploit relatively higher level information from data streams, we need to combine data streams from different aspects, as well as static background knowledge together, and use querying or reasoning process to get the answers of interest. However, due to the heterogeneous nature of streaming information, it is essential to represent the data streams and static knowledge with a unified, unambiguous semantic model. On the other hand, the streaming information is temporal. Compared with static knowledge, every temporal statement is only valid during a short time interval, the different valid time results in the temporal correlations among them are extremely complex, and if not carefully modelled, will lead to completely different meanings. Briefly, to capture and exploit the semantics carried by streaming information, we need:

- Represent and integrate both data streams and background knowledge with an unambiguous data model.
- Query for not only contents of streaming information, but also the temporal correlations among them.

On the data processing model, one-time query and batch-processing model cannot capture the dynamic semantics of streaming information. Indeed, we need a continuous query execution plan, as well as a dynamic management strategy for data stream. Also, the large amount and high updating frequency of streaming information determines the general data storing and index strategy will no longer be useful. So to sum up, to cope with streaming information on the data processing level, we need:

- A dynamic query execution plan for continuous queries over data streams.
- An effective and extensible management plan for large amount and frequently-updating data streams.

Therefore, in this paper we propose CT-SPARQL and AIMRS, which are query language specifying for temporal RDF stream patterns, and the system for adaptive incremental maintenance of RDF streams.

*Supervised by Xiaowang Zhang.

WWW '18 Companion, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5640-4/18/04.
https://doi.org/10.1145/3184558.3186573

2 STATE OF THE ART

The processing model of large amounts of streaming information has been thoroughly studied in [1]. Recently with the development of Semantic Web, there is a trend to lift streaming information into semantic level, and combine them with background knowledge so that streaming information of different aspects can interact with each other [2]. Therefore, efforts have been made to extend both RDF [7, 8] and SPARQL [9, 10], which are W3C recommended meta data model for representing semantic information on the Web and standard RDF query language, for stream tuples and continuous querying functionalities.

On the meta data model level, RDF is designed for representing persistent relations among resources on the Web. To leverage the data model of RDF to represent streaming, temporal information, one problem is how to encode the time annotations of RDF statements into the data model of RDF. Existing works [5, 6] have studied this problem, and show that both time-point-based and time-interval-based representation are equivalent for representing time-annotated RDF statements. On the other hand, as it is studied in [21], the temporal correlations among the valid-time-intervals of temporal statements are of great significance, and need to be precisely captured and modeled. Unfortunately, existing RDF stream processing languages such as C-SPARQL [11, 12], CQELS [13] and EP-SPARQL [14], none of these languages is able of querying for all the temporal relations listed in [21]. C-SPARQL and CQLES, due to the reliance on the semantics of CQL, is unable to detect the fine-grained temporal relations within window. EP-SPARQL, though enriches every temporal RDF statement with an time interval, however, the temporal relations in EP-SPARQL are limited to the relations among time points, and therefore cannot model the temporal relations among time intervals.

On the processing model level, the Continuous Query Language (CQL) [3, 4], as the firstly proposed language for registering continuous queries over updatable relations, has thoroughly studied and classified the operations in stream processing, into three kinds. Namely, the stream-to-relation (S2R) operation, relation-to-relation (R2R) operation, and relation-to-stream (R2S) operation. Existing RDF stream processing systems are all based on the processing model of CQL, only extending R2R operation for SPARQL queries in different ways. C-SPARQL integrates Esper and Jena [24], which are two engines for CQL and SPARQL to realize C-SPARQL engine, however, this approach also limits its performance and potential for optimization. CQELS trades query complexity for efficiency, and realizes a special RDF encoding, a native implementation for SPARQL operations such as AND, UNION and FILTER over these specially encoded RDF data, and most importantly, an adaptive mapping strategy according to real-time processing conditions, to achieve high system performance. EP-SPARQL realizes its system on top of ETALIS [15], which is a system implemented in Prolog. Though EP-SPARQL is able to capture the relatively more complex temporal correlations among RDF stream tuples than C-SPARQL and CQELS, its implementation is still based on the batch-processing model, thus lacks of scalability in stream processing scenario. PRSP [16–18] presents a plugin-based framework for RDF stream processing, and enables to integrate different kinds of RDF storage engines together, to execute querying and reasoning process over large amounts of RDF streams. Strider [19] presents a hybrid adaptive distributed RDF stream processing engine, which can process large amounts of RDF data by using Spark [20].

3 PROPOSED APPROACH

We believe that RDF stream, namely the time-annotated RDF statement is the suitable data model for representing streaming information on the Web. However, our approach differs from the existing works mainly in two aspects, the query language and the processing model.

3.1 Query Language

Existing RDF stream processing languages such as C-SPARQL and CQELS, are the extensions of CQL and SPARQL. However, the adoption of window has limited their expressive power. Patterns defined in these languages must fall in a window, which is a set of pre-defined time intervals, otherwise they will not be detected. However, this limitation is not explicitly defined in the pattern itself. In fact, most of the data streams are unpredictable, and we cannot pre-define a window that covers all the stream tuples that satisfy the temporal patterns. We believe that when defining temporal patterns, the user should focus on the temporal correlations among stream tuples, instead of the implicit temporal restrictions caused by windows. Therefore, the first difference of our continuous query language is the elimination of window and the clear semantics of continuous query that is independent of window.

Secondly, when dealing with the complex temporal correlations among RDF stream tuples, the drawback of existing query languages mainly lies in:

- Cannot describe the negation of stream tuples in a time interval.
- Cannot decide which to choose when a sequence of stream tuples satisfy the pattern in multiple ways.

Therefore, we introduce a new binary operator for defining the unique time-sequential relation among two stream patterns, and show how to use this operator to define compositional stream patterns. We believe in this way we are able to define any kind of complex temporal correlations among stream tuples, and precisely choose the stream tuples of interest no matter how their time-sequential order is.

3.2 Processing Model

Existing RDF stream processing systems such as C-SPARQL, CQELS and EP-SPARQL are based on the processing model of CQL. That is, use stream-to-relation operators, which are windows, to extract portions of stream tuples, and use relation-to-relation operators, which are SPARQL operators,

to execute queries over these portions, and finally use relation-to-stream operators to stream out the answers. However, these approaches have two fatal drawbacks. First, the portions of RDF stream tuples that are extracted with windows, are directly transformed into one single RDF graph, without taking into consideration of their different time annotations, which makes it impossible to exploit temporal information from this RDF graph. Second, existing RDF stream processing systems still use engines such as gStore [22], RDF-3X [23] and Jena [24], to execute queries over RDF stream tuples. These engines are designed for executing queries multiple times over a batch of persistent data. Before query execution, these engines will build complex index over the data to achieve high query execution performance, which is also a time-consuming process. However, in the CQL processing model, continuous queries registered over window only need be executed per update, and the contents of window is updated in high frequency. Therefore, it is unnecessary and a waste of system resources to build complex index over the contents of window, and relying on the query execution plan of engines designed for persistent data will inevitably lead to low system performance and bad scalability.

We believe that split RDF stream into several snapshots is better than maintaining a total version of RDF stream tuples in a pre-defined window. First, we can gather RDF stream tuples of different time annotations into a set of snapshots, every snapshot is represented with a unique RDF graph, and thus the temporal information can be preserved as a set of temporal relations among these named graphs(snapshots). Second, building indexes over a set of smaller RDF graphs is lesser time-consuming, and more importantly, it can be processed concurrently. Last but not least, in this storage strategy, a query over multiple snapshots will be decomposed into a set of sub-queries over single snapshot. And by joining answers of sub-queries step-by-step, we can get the equivalent answers of the original query. Therefore, this approach suits more the real-time processing feature of streaming information. We can dynamically modify the sub-query execution plan according to the intermediate results, and furthermore, delete the expired intermediate results in time to reduce system overhead.

4 METHODOLOGY

In this section we introduce the distinguishing features of our approach in detail, and demonstrate part of our on-going works.

4.1 Compositional Temporal Relations

We introduce our approach for defining compositional temporal relations among stream tuples. First, assume two disjoint sets I, T, representing the instantaneous event type and time point domain, respectively, and the data stream S is a set of tuples from $I \times T : S = \{(i,t)|i \in I, t \in T\}$. An instantaneous event expression i is a function from time point domain onto boolean values: $i : T \rightarrow \{True, False\}$. And this function evaluates to true if and only if there is an instantaneous event of type i occurs at time point t: $i(t) \equiv (i,t) \in S$.

We then demonstrate the syntax and semantics of a basic temporal relation: SEQ. Informally, SEQ claims the *nearest* time sequential relation from one event type to another. Take a sequence of instantaneous events:
$\{(A_1,t_1),(A_2,t_2),(B_1,t_3),(B_2,t_4)\}$ as example, the relation A SEQ B is a set of tuples:
$\{((A_1,t_1),(B_1,t_3)),((A_2,t_2),(B_1,t_3))\}$. Formally, we have:
i_1 SEQ $i_2 = \{((i_1,t_1),(i_2,t_2))|\forall(i_2,t') \in S.t' > t_1 \rightarrow t' - t_1 \geq t_2 - t_1\}$.

In fact, the temporal relations among stream tuples are compositional, and we believe that any complex temporal relations can be decomposed into a set of basic SEQ relations. Further, we give the formal syntax of compositional event type.
(1) An instantaneous event type i from I is a compositional event type.
(2) If i_1, i_2 are instantaneous event types, then expression (i_1 SEQ i_2) is a compositional event type.
(3) If c_1, c_2 are compositional event types, then expressions $(\neg c_1),(c_1 \wedge c_2),(c_1 \vee c_2)$ are compositional event types.

The discussion of the expressivity of SEQ relation will be our future work, however, we here demonstrate the expressive power of compositional events defined by SEQ relation through two examples. Given the example data stream:
$S = \{(A_1,t_1),(C,t_2),(B_1,t_3),(A_2,t_4),(B_2,t_5)\}$

(1) Negation of event.
Assume we want to find out the compositional events that start with an event of type A, end with an event of type B, and without having any events of type C during the time interval (A,B). This compositional event can be expressed as expression: (A SEQ B) $\wedge$ (A SEQ ($B \vee C$)). The conjunction of event (A SEQ B) and event (A SEQ ($B \vee C$)) guarantees that no event of type C occurs during the interval (A,B), and the mapping set of this event expression over S is $\{((A_2,t_4),(B_2,t_5))\}$.

(2) Containment of event.
Assume we want to find out the events that start with an event of type A, end with an event of type B, and with at least one event of type C during the interval (A,B). This compositional event can be expressed as expression:
$((A \text{ SEQ } (B \vee C)) \wedge (\neg(A \text{ SEQ } B))$. And the mapping set of this event expression over S is $\{((A_1,t_1),(B_1,t_3))\}$.

In this way, the two kinds of time intervals can be explicitly distinguished, while existing RDF stream processing languages such C-SPARQL, CQELS and EP-SPARQL cannot.

4.2 CT-SPARQL

In this section we introduce Continuous Temporal SPARQL (CT-SPARQL), which is a query language extending standard SPARQL syntax with SEQ relation. And we show how to combine standard SPARQL graph pattern with SEQ relation to form *RDF stream pattern*.

Assume three mutually disjoint sets U, B, L, representing the URI set, Blank Node set, and Literal set, respectively. An RDF triple is a triple (s, p, o) from $(U \cup B) \times U \times (U \cup B \cup L)$. Assume additionally a set V that is disjoint with $U \cup B \cup L$, representing the variable set.

An RDF stream pattern is recursively defined as follows:
(1) If P is an RDF graph pattern, then P is an RDF stream pattern.
(2) If P_1, P_2 are RDF stream patterns, then expression (P_1 SEQ P_2) is an RDF stream pattern.
(3) If P_1, P_2 are RDF stream patterns, then expressions $(\neg P_1)$, $(P_1 \wedge P_2)$, and $(P_1 \vee P_2)$ are RDF stream patterns.

And the SPARQL graph pattern is recursively defined as follows:
(1) A tuple from $(U \cup L \cup V) \times (U \cup V) \times (U \cup L \cup V)$ is an RDF graph pattern.
(2) If P_1, P_2 are RDF graph patterns, then the expressions (P_1 AND P_2), (P_1 UNION P_2), (P_1 OPT P_2) are RDF graph patterns.
(3) If P is an RDF graph pattern and R is a SPARQL built-in condition, then the expression (P FILTER R) is a graph pattern.

In CT-SPARQL, the instantaneous event is represented with SPARQL graph pattern, and the compositional event is represented with stream pattern, which is a set of graph patterns connected with SEQ relations. This approach combines the SPARQL and SEQ relation together, so that CT-SPARQL can exploit the expressive power of SPARQL for describing the heterogeneous relations between streaming information and background knowledge, as well as the compositional temporal relations defined on top of SEQ relation, to unambiguously query for RDF streams.

4.3 AIMRS

In this section we introduce the architecture of Adaptive Incremental Maintenance of RDF Stream (AIMRS). The architecture of AIMRS is shown in Figure 1, and it is a system designed for executing continuous queries of CT-SPARQL over large volumes of RDF streams. We next briefly illustrate the important components of AIMRS, and how they cooperate with each other.

4.3.1 Query Parser. The query parser divides a query over multiple snapshots into a set of sub-queries over single snapshot. For example, a CT-SPARQL query Q:
{?vehichle :registeredAt :roadA}
SEQ
{?vehicle :registeredAt :roadB}
SEQ
{?vehicle :registeredAt :roadC}
is divided to a set of sub-queries:
Q'={Q_1 :{?vehicle :registeredAt :roadA},
Q_2 :{?vehicle :registeredAt :roadB},
Q_3 :{?vehicle :registeredAt :roadC}}

And this set of sub-queries Q' will be used to generate a dynamic query plan according to the former query execution results.

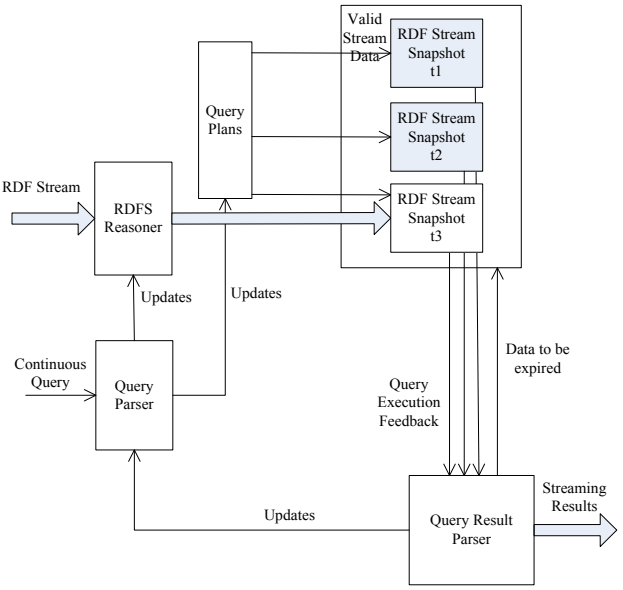

Figure 1: Architecture of AIMRS

4.3.2 Dynamic Query Plan. The basic strategy of dynamic query plan is, to remove the expired stream tuples from memory as early as possible, since the sub-queries are connected with SEQ relations, this can be realized. Another point is that, a sub-query is executed only when its predecessor sub-query has results in the last snapshot. For example, Q_1 detects stream tuple (:vehicle :registeredAt :roadA) at snapshot 1, then in snapshot 2 both Q_1 and Q_2 will be executed. And once Q_2 detects stream tuple (:vehicle :registeredAt :roadB) at any snapshot, the stream tuple (:vehicle :registeredAt :roadA) in snapshot1 will be deleted.

4.3.3 Dynamic Reasoning Process. To ensure that streaming information from heterogeneous data sources can interact with each other on the semantic level, it is necessary to enrich them with background knowledge. For example, the streaming information received from GPS sensor is a set of longitude and latitude values such as (:vehicle :hasLongitude :literal1) and (:vehicle :hasLatitude :literal2). And the background knowledge relates these geographical values with roads such as (:roadA :hasLongitude :literal1) and (:roadA :hasLatitude :literal2). In this case we will use the reasoning process to add another triple (:vehicle :registeredAt :roadA) for further querying process.

However, the reasoning process is time-consuming, and we do not need to reason for all the rules at every snapshot. Therefore the reasoning process should also be dynamic. The rules to be used for enriching RDF streams depends on the dynamic query execution plan. For example, if Q_1 and Q_2 will be executed in the next snapshot, then only the rules relevant to :roadA and :roadB need to be used for reasoning process.

5 RESULTS

The evaluation of AIMRS will mainly include two criterions: the correctness of CT-SPARQL queries and the system performance.

5.1 Correctness

The barriers ahead of the validation of CT-SPARQL is twofold. First, our on-going works haven't come to the semantic level of CT-SPARQL. Though we are able to define complex stream patterns, but we haven't found out its connections with existing RDF stream processing languages such as C-SPARQL, CQELS and EP-SPARQL. Second, currently there lacks a way for customized generating RDF stream data of interest. Though existing works have provided RDF stream data and testing queries from C-SPARQL, CQELS, and EP-SPARQL. However, these works are focused on the semantics of CQL, i.e. continuous querying functionalities, and the temporal correlations among RDF stream tuples is not concerned.

Therefore, our evaluation work will mainly include two parts:
(1) The analysis for the semantics of CT-SPARQL language.
(2) The customized way for generating RDF stream tuples with complex temporal correlations.

5.2 System Performance

This part of evaluation mainly includes two parts: the throughput and query-answering delay. The throughput decides how big RDF stream processing scenario that AIMRS can be applied to, and the query-answering delay decides how many kinds of scenarios that AIMRS can be applied to.

6 CONCLUSIONS AND FUTURE WORKS

In this paper, we analyze the status of existing RDF stream processing works, and point out a new direction: the semantics and processing model for exploiting the complex temporal correlations among RDF stream tuples. We believe in real-time applications such as traffic monitoring, sensor networks, and social media, these fine-grained relations, once get appropriately explored, will greatly enrich the streaming information, and motivate a wide range of time-sensitive applications.

As we have proposed the syntax of CT-SPARQL and architecture of AIMRS, our future work will focus on the formalization of CT-SPARQL and the implementation of AIMRS.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (61672377) and the National Key Research and Development Program of China (2016YFB1000603).

REFERENCES

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. *Models and issues in data stream systems.* ACM Sigmod-Sigact-Sigart Symposium on Principles of Database Systems, 1-16, 2002.
[2] A. Margara, J. Urbani, F. V. Harmelen, and H. Bal. *Streaming the Web: Reasoning over dynamic data.* Web Semantics Science Services Agents on the World Wide Web, 25(1):24-44, 2014.
[3] A. Arasu, S. Babu, and J. Widom. *CQL: A Language for Continuous Queries over Streams and Relations.* Lecture Notes in Computer Science, September, 2003.
[4] A. Arasu, S. Babu, and J. Widom. *The CQL continuous query language: semantic foundations and query execution.* The VLDB Journal, 15(2):121-142, 2006.
[5] C. Gutierrez, C. Hurtado, and A. Vaisman. *Temporal RDF.* European Semantic Web Conference, 93-107, 2005.
[6] C. Gutierrez, C. Hurtado, and A. Vaisman. *Introducing time into RDF.* IEEE Transactions on Knowledge and Data Engineering, 19(2):207-218, 2007.
[7] Hayes, Patrick, McBride, and Brian. *RDF Semantics 1.1.* W3C Proposed Recommendation, 2014.
[8] R. Cyganiak, D. Wood, and M. Lanthaler. *RDF 1.1 Concepts and Abstract Syntax.* W3C Proposed Recommendation, 2014.
[9] C. Gutierrez, C. Hurtado, and A. Mendelzon. *Formal aspects of querying RDF databases.* Proc. of SWDB, 293-307, 2003.
[10] S. Harris and A. Seaborne. *SPARQL 1.1 Query Language.* W3C Proposed Recommendation, 2013.
[11] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus. *C-SPARQL: SPARQL for continuous querying.* International Conference on World Wide Web, 1061-1062, 2009.
[12] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, and M. Grossniklaus. *Querying RDF streams with C-SPARQL.* ACM SIGMOD Record, 39(1), 20-26, 2010.
[13] D. Le-Phuoc, M. Dao-Tran, J. X. Parreira, and M. Hauswirth. *A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data.* International Semantic Web Conference, 370-388, 2011.
[14] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic. *EP-SPARQL: a unified language for event processing and stream reansoning.* International Conference on World Wide Web, 635-644, 2011.
[15] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic. *Etalis: Rule-based reasoning in event processing.* Reasoning in Event-based Distributed Systems, Studies in Computational Intelligence seeries. LNCS, Springer Verlag, 2011.
[16] Q. Li, X. Zhang, and Z. Feng. *PRSP: A Plugin-based Framework for RDF Stream Processing.* International Conference on World Wide Web, 815-816, 2017.
[17] Q. Li, X. Zhang, and Z. Feng. *An Adaptive Framework for RDF Stream Processing.* Asia-pacific Web, 427-443, 2017.
[18] Q. Li, X. Zhang, Z. Feng, and G. Xiao. *An Adaptive Framework for RDF Stream Reasoning.* International Semantic Web Conference, Posters and Demos, 2017.
[19] X. Ren and O. Cure. *Strider: A Hybrid Adaptive Distributed RDF Stream Processing Engine.* International Semantic Web Conference, 559-576, 2017.
[20] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. *Spark: cluster computing with working sets.* Usenix Conference on Hot Topics in Cloud Computing, 15(1): 10-10, 2010.
[21] J. F. Allen. *Maintaining Knowledge about Tempoarl Intervals.* Readings in Qualitative Reasoning About Physical Systems, 26(11):361-372, 1983.
[22] L. Zou, M. T. Ozsu, L. Chen, X. Shen, R. Huang, and D. Zhao. *gStore: a graph-based SPARQL query engine.* VLDB Journal, 23(4):565-590, 2014.
[23] T. Neumann and G. Weikum. *The RDF-3X engine for scalable managment of RDF data.* VLDB Journal, 19(1):91-113, 2010.
[24] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. *Jena: implementing the semantic web recommendations.* International World Wide Web Conference, 16:74-83, 2004.