Erschienen in: Proceedings of the 2nd Workshop on Social Data on the Web (SDoW2009), collocated with the 8th International Semantic Web Conference (ISWC-2009) / Breslin, Urs et al. (Hrsg.). - (CEUR Workshop Proceedings ; 520)

Continuous Queries and Real-time Analysis of Social Semantic Data with C-SPARQL

Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus

Dipartimento di Elettronica e Informazione, Politecnico di Milano
Piazza Leonardo da Vinci 32, I-20133 Milano, Italy
{dbarbieri|braga|ceri|dellavalle|grossniklaus}@elet.polimi.it

Abstract. Social semantic data are becoming a reality, but apparently their streaming nature has been ignored so far. Streams, being unbounded sequences of time-varying data elements, should not be treated as persistent data to be stored "forever" and queried on demand, but rather as transient data to be consumed on the fly by queries which are registered once and for all and keep analyzing such streams, producing answers triggered by the streaming data and not by explicit invocation. In this paper, we propose an approach to continuous queries and real-time analysis of social semantic data with C-SPARQL, an extension of SPARQL for querying RDF streams.

Keywords: Continuous SPARQL, Social Semantic Data, Continuous Query, Real-time Analysis

1 Introduction and Motivation

"Which are the hottest topics under discussion on Twitter?" "Who is discussing about Italian food right now?" "What have my close friends been discussing in the last hour?" "Who is now discussing about Tuscany red wines in my social network?" "How many people have been twittering about northern Italy white wines in the last three hours?"

The information required to answer those queries is increasingly becoming available on the Web. On the one side, we observe a trend in Web 2.0 as blogs, feeds and microblogs are adopted to disseminate and publish information in real-time streams trough social networking Web sites. This trend is often referred to as the Twitter phenomenon or, in more broader terms, as the so-called blogosphere. On the other side, a trend can be also observed towards the interlinking of Social Web with semantics [1] using vocabularies such as Semantically-Interlinked Online Communities[1] (SIOC), Friend-of-a-Friend[2] (FOAF) and Simple Knowledge Organization System[3] (SKOS).

[1] http://rdfs.org/sioc/spec/
[2] http://xmlns.com/foaf/spec/
[3] www.w3.org/2004/02/skos/

As concrete examples, we can refer to three pioneers in this field

- SMOB [2] is a distributed and decentralized microblogging system built on SIOC and FOAF, for which the authors implemented both a publishing and an aggregating service prototype.
- Smesher [3] is a Semantic Microblogging client that integrates Twitter and identi.ca, detects structure in microposts, extracts content in RDF and allows SPARQL queries over the extracted information.
- SemanticTweet[4] is a simple Web service that generates a FOAF RDF document from the list of Twitter friends and followers of any Twitter user using the Twitter REST API.

Several attempts to provide answers to those questions exist both by employing Information Retrieval (IR) methods (e.g., technorati.com, icerocket.com, blogsearchengine.com, blogsearch.google.com) and Semantic Web methods (e.g., the SIOC API of sindice.com, SMOB aggregator and Smesher). Our claim is that answering such questions in the space of *one-time semantics* of current IR and Semantic Web tools is difficult because the underlying data are streams.

Data streams are unbounded sequences of time-varying data elements. They have been recognized in a variety of modern applications, such as network monitoring, traffic engineering, sensor networks, RFID tag applications, telecom call records and financial applications. Processing of data streams has been largely investigated in the last decade [4], specialized Data Stream Management Systems (DSMS) have been developed, and features of DSMS are becoming supported by major database products, such as Oracle and DB2.

DSMS represent *a paradigm change* in the database world as they move from persistent relations and user-invoked queries to transient streams and *continuous queries*. The innovative assumption is that streams can be *consumed* on the fly rather than being stored forever and that queries which are persistently monitoring streams are able to produce their answers even in the absence of invocation. DSMS can support parallel query answering over data originating in real-time and can cope with bursts of data by adapting their behavior to gracefully degrade answer accuracy by introducing higher levels of approximation. However, even if such DSMS systems proved to be an optimal solution for on the fly analysis of data streams, they cannot perform complex reasoning tasks, such as the ones required for computing the answers to the above questions.

At the same time, while Semantic Web reasoners are year after year scaling up in the classical, time invariant domain of RDF triples and ontological knowledge, reasoning upon rapidly changing information has been neglected or forgotten so far. Reasoning systems assume static knowledge, and do not manage "changing worlds"—at most, one can update the ontological knowledge and then repeat the reasoning tasks.

In [5], we propose Stream Reasoning as the new multi-disciplinary approach which will provide the abstractions, foundations, methods, and tools required to

[4] http://semantictweet.com/

integrate data streams and reasoning systems, thus giving answer to the above and other questions from different domains.

The rest of the paper is organized as follows. In Section 2, we provide the background needed to understand the proposed extensions to RDF and SPARQL introduced in Sections 3.1 and 3.3, respectively. In Section 4, we describe an architecture and implementation of a C-SPARQL engine. Section 5 is dedicated to the comparison of C-SPARQL to SPARQL using the real-world social data streams described in Section 3.2. We close the paper by discussing related work in Section 6 and draw conclusions in Section 7.

2 Background

This section illustrates previous work on data streams and the SPARQL language.

2.1 Data Streams

DSMS are based on the observation that not only is it impossible to control the order in which data items arrive in a stream, but, even more importantly, it is not feasible to locally store a stream in its entirety [6].

The Chronicle data model [7] is one of the first models proposed for data streams. It introduced the concept of chronicles, append-only ordered sequences of tuples, as well as a restricted view definition language and an algebra that operates both over chronicles and traditional relations. OpenCQ [8], NiagaraCQ [9] and Aurora [10] are representative implementations of DSMS addressing continuous queries and distribution issues.

The first query language tailored to data streams, CQL [11, 12], was the result of research done by Babu et al. [13] on the problem of continuous queries over data streams, addressing semantic issues as well as efficiency concerns. They specify a general and flexible architecture for query processing in the presence of data streams.

More recently, Law et al. [14] put particular emphasis on the problem of *mining* data streams [15]. They conceived and developed Stream Mill [16], which considers and addresses data mining issues extensively, specifically with respect to the problem of online data aggregation and to the distinguishing notion of blocking and non-blocking operators. Its query language (ESL) efficiently supports physical and logical windows (with optional slides and tumbles) on both built-in aggregates and user-defined aggregates. The constructs introduced in ESL extend the power and generality of DSMS.

2.2 SPARQL

SPARQL has been developed under the patronage of the W3C as the standard query language for RDF. Therefore, the most authoritative source on its syntax

and semantics is the W3C recommendation [17]. Several papers, however, discuss extensions to the SPARQL language as defined by the W3C.

With the goal of proposing a syntactic and semantic extension to SPARQL, we found the following works particularly useful.

- Gutierrez et al. [18], who define a conjunctive query language for RDF with basic patterns, which is a formal and unambiguous basis for defining the semantics of SPARQL queries evaluation;
- Perez et al. [19], who analyze the semantics and complexity of SPARQL; and
- Cyganiak [20] and Haase et al. [21], who independently present a relational model of SPARQL which allows to implement SPARQL queries over a relational database engine.

3 RDF Streams and Continuous SPARQL

Data models and query languages for DSMS are not sufficient for continuously querying and analyzing in real-time streams of RDF. Indeed, we deem that there is a potential interest for giving up *one-time semantics* in RDF repositories as well, so as to explore the benefits provided by *continuous semantics*. Therefore, we introduce *RDF streams* as the natural extension of the RDF data model to the new continuous scenario and Continuous SPARQL (or simply *C-SPARQL*) as the extension of SPARQL for querying RDF streams.

3.1 RDF Streams

An RDF stream is defined as an ordered sequence of pairs, where each pair is constituted by an RDF triple and its timestamp τ.

$$\begin{gathered}\dots\\(\langle subj_i, pred_i, obj_i\rangle\ ,\ \tau_i)\\(\langle subj_{i+1}, pred_{i+1}, obj_{i+1}\rangle\ ,\ \tau_{i+1})\\\dots\end{gathered}$$

Timestamps can be considered the *context* of RDF triples. They are monotonically non-decreasing in the stream ($\tau_i \leq \tau_{i+1}$). They are not strictly increasing because timestamps are not required to be unique. Any (unbounded, though finite) number of consecutive triples can have the same timestamp, meaning that they *occur* at the same time, although sequenced in the stream according to some positional order. Our definition of RDF streams extends RDF in the same way as the stream type in CQL extends the relation type.

Named graphs [22] and N-Quads [23], a format that extends N-Triples with context, can be both adopted as a concrete serialization for RDF streams. For our experiments we adopt N-Quads and we use as context the timestamp encoded as a RDF literal of type `xsd:dateTime`.

3.2 A Real Source of Social Semantic Data Streams

Given that RDF streams of social semantic data are not readily available yet, we decided to use the data provided by Social Network Glue[5]. Glue enables users to connect with their friends on the Web based on the pages the users visit online. Using semantic recognition technologies to automatically identify books, music, movies, wines, stocks, movie stars and many other similar topics, it generates a continuous stream of the identified objects which is accessible in real time using a REST API[6]. The REST request "http://api.getglue.com/v1/glue/recent" returns the 250 most recent public interactions. We adopted the GRDDL approach [24] and implemented a simple way to translate the resulting XML into RDF. A live version is running at http://c-sparql.cefriel.it/sdow-demo. Below, we provide a snapshot of the resulting RDF stream.

Subject	Predicate	Object	Timestamp
glueinter:i1	rdf:type	sioc:Post	"2009-07-20T22:48:52Z"
glueinter:i1	sioc:content	"The Proposal on imdb.com"	"2009-07-20T22:48:52Z"
glueinter:i1	sioc:has_container	http://www.getglue.com	"2009-07-20T22:48:52Z"
glueinter:i1	dc:title	"The Proposal"	"2009-07-20T22:48:52Z"
glueuser:id1	rdf:type	sioc:User	"2009-07-20T22:48:52Z"
glueinter:i1	sioc:has_creator	glueuser:id1	"2009-07-20T22:48:52Z"
glueinter:i1	sioc:topic	gluecat:movies	"2009-07-20T22:48:52Z"
glueinter:i2	rdf:type	sioc:Post	"2009-07-20T22:48:54Z"
glueinter:i2	sioc:content	"Mario Kart Wii on gamefaqs.com"	"2009-07-20T22:48:55Z"
glueinter:i2	sioc:has_container	http://www.getglue.com	"2009-07-20T22:48:55Z"
glueinter:i2	dc:title	"Mario Kart Wii"	"2009-07-20T22:48:55Z"
glueuser:id2	rdf:type	sioc:User	"2009-07-20T22:48:55Z"
glueinter:i2	sioc:has_creator	glueuser:id2	"2009-07-20T22:48:55Z"
glueinter:i2	sioc:topic	glueint:video_games	"2009-07-20T22:48:55Z"

Similarly to SemanticTweet, we are also able to translate the social relationships obtained using the REST service "/user/friends" into FOAF.

3.3 Continuous SPARQL

C-SPARQL is an extension of SPARQL for querying both RDF graphs and RDF streams. The complete definition of the language in terms of syntax and semantics is given in [25]. We briefly repeat the definitions of the distinguishing features of the language here and show how to write the queries that evaluate the answers to the questions which opened this paper.

Continuous Queries The distinguishing feature of C-SPARQL is the support for continuous queries, i.e. queries that are *registered* and then executed continuously over *windows* opened on RDF streams and standard RDF graphs. Continuous queries, which make usage of *aggregates*, are particularly relevant.

A C-SPARQL query is *registered* using the grammar extension provided by the first of the following two production rules.

[5] http://getglue.com
[6] http://getglue.com/api

Registration → 'REGISTER QUERY' *QueryName* 'AS' *Query*
Registration → 'REGISTER STREAM' *QueryName* 'AS' *Query*

As output C-SPARQL queries produce the same types as SPARQL queries: boolean answers, selections of variable bindings, RDF descriptions of the involved resources or constructions of new RDF triples. These outputs are continuously renewed in each query execution. In addition, C-SPARQL queries can be registered to produce new RDF streams using the grammar extension provided by the second production rule given above. In this second case, only CONSTRUCT and DESCRIBE queries can be registered, as they produce RDF triples that, once associated with a timestamp, yield RDF streams that can be managed in C-SPARQL.

Windows Given that RDF streams are intrinsically infinite, we introduce the notion of *windows* upon RDF streams, whose types and characteristics are inspired by those of the windows in continuous query languages such as CQL [12]. Identification and windowing are expressed in C-SPARQL by means of the FROM STREAM clause.

FromStrClause → 'FROM' ['NAMED'] 'STREAM' *StreamIRI* '[RANGE' *Window* ']'

From the RDF stream identified by StreamIRI, a window extracts the *last* triples, which are considered by the query. The extraction can be *physical* (a given number of triples) or *logical* (a variable number of triples which occur during a given time interval).

The part of C-SPARQL that we introduced so far is sufficient to address the question "What have my closest friends been visiting in the last hour?" Below, we show how to formulate this query in C-SPARQL over the Glue interaction stream and the graph of FOAF relationships described in Section 3.2.

```
REGISTER QUERY WhatHaveMyCloseFriendsBeenVisitingInTheLastHour AS
PREFIX sioc: <http://rdfs.org/sioc/ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX glue: <http://c-sparql.cefriel.it/sdow-demo/>
SELECT DISTINCT ?friend ?topic
FROM <http://c-sparql.cefriel.it/sdow-demo/glueusers.rdf>
FROM STREAM <http://c-sparql.cefriel.it/sdow-demo/interactions.trdf>
     [ RANGE 60m STEP 5m ]
WHERE { glue:id1 foaf:knows ?friend .
        ?post    sioc:has_creator ?friend .
        ?post    rdf:type sioc:Post .
        ?post    sioc:topic ?topic . }
```

The first triple pattern matches triples in the FOAF graph, whereas the other three triple patterns match triples in a sliding window of 60 minutes opened on the RDF stream, which advances progressively in steps of 5 minutes.

Aggregation Another question that we could answer using a C-SPARQL is "Which are the top topics in Glue?" To do so, we also need to introduce the

aggregation capabilities that we added to C-SPARQL. We allow multiple independent aggregations within the same C-SPARQL query, thus pushing the aggregation capabilities beyond those of SQL and other proposals for aggregation in SPARQL[7]. Aggregation clauses have the following syntax.

AggregateClause → ('AGGREGATE { (' *var* ',' *Fun* ',' *Group* ')' [*Filter*] '}')*
Fun → 'COUNT' | 'SUM' | 'AVG' | 'MIN' | 'MAX'
Group → *var* | '{' *var* (',' *var*)* '}'

An aggregation clause starts with a new variable not occurring in the WHERE clause, followed by an aggregation function and closed by a set of one or more variables, occurring in the WHERE clause, which express the grouping criteria. For instance, the query above can be expressed in C-SPARQL as follows.

```
REGISTER QUERY TopTopicsGlueUsersAreInterestedIn AS
PREFIX sioc: <http://rdfs.org/sioc/ns#>
SELECT DISTINCT ?topic ?number
FROM STREAM <http://c-sparql.cefriel.it/sdow-demo/glueinteractions.trdf>
     [ RANGE 30m STEP 10m ]
WHERE { ?post sioc:topic ?topic . }
AGGREGATE {(?number , COUNT , {?topic })}
```

4 A C-SPARQL Engine

The C-SPARQL engine was designed based on a separation of concerns between stream management and query evaluation. This separation is the foundation for a simple architecture for C-SPARQL, built upon known database and reasoning technolgies. Figure 1 shows the three main components of our C-SPARQL execution framework.

The module named *C-SPARQL Query Parser* gets a C-SPARQL query as input and produces the information needed by the *Data Stream Manager Layer* and the *SPARQL EndPoint Layer* to execute the query. The data stream manager registers the data streams specified in the query and applies logical or physical windows. When the resulting graph has been produced, the SPARQL part of the C-SPARQL query is executed by the SPARQL endpoint. This process is executed as frequently as specified in the REGISTER clause of the C-SPARQL query. Finally, the result computed is timestamped and passed on. Both the data stream manager and the SPARQL endpoint are considered plugins, in order to be independent from the actual DSMS/SPARQL engine implementations that will be used. We have implemented a prototype based on this architecture using ESPER as a DSMS and Jena as a SPARQL endpoint.

ESPER[8] is a component for stream processing applications, which require high throughput to process large volumes of data elements (between 1,000 to 100k messages per second) and low latency to react in real-time (from a few

[7] http://esw.w3.org/topic/SPARQL/Extensions/Aggregates
[8] http://esper.codehaus.org/

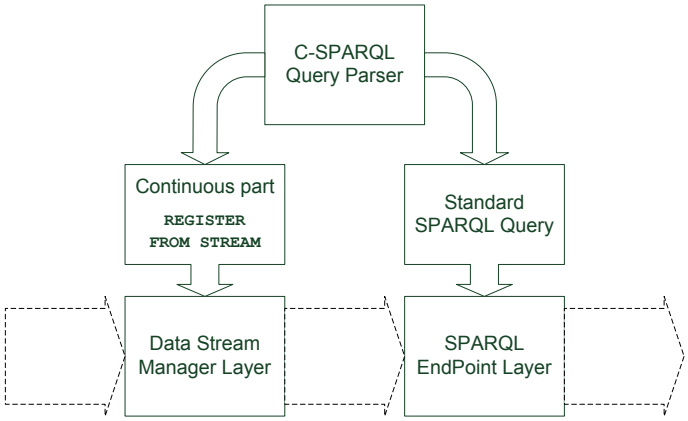

Fig. 1. C-SPARQL Engine Architecture

milliseconds to a few seconds). In particular, it supports the various forms of windows we defined in C-SPARQL (i.e., sliding and tumbling windows both in terms of time and length) and several forms of aggregation we plan to exploit for optimizations based on query rewriting.

Jena[9] is a Java framework for building Semantic Web applications. We choose it, because it includes a custom RDF storage engine for high performance applications and a SPARQL query engine. The SPARQL engine supports standard SPARQL and aggregation, GROUP BY and assignment as SPARQL extensions.

The adoption of off-the-shelf stream management systems and reasoning tools both provide a solid framework and a fast way of prototyping.

5 Evaluation

In order to evaluate, our approach we compared the time required to compute a C-SPARQL query with our engine, to the time needed to execute an equivalent SPARQL query in Jena using its SPARQL engine over its custom RDF storage engine. The tests have been run on a Pentium Core 2 Quad 2.0GHz with 2GB of main memory.

As representative C-SPARQL query we chose the following one, in which we count how many Glue users are interested in the various topics recognized by Glue in the last 3 minutes.

```
REGISTER QUERY CountHowManyGlueUsersAreInterestedInEachTopic AS
PREFIX sioc: <http://rdfs.org/sioc/ns#>
SELECT DISTINCT ?topic ?number
FROM STREAM <http://c-sparql.cefriel.it/sdow-demo/interactions.trdf>
     [ RANGE 3m STEP 10s ]
WHERE { ?post sioc:topic ?topic . }
AGGREGATE {(?number , COUNT , {?topic })}
```

[9] http://jena.sourceforge.net/

This simple query showcases all characteristics of C-SPARQL, namely, registration, selection of triples based on a window over a RDF stream and an aggregate function.

We registered this C-SPARQL query in our engine and continuously executed it every second. We run two experiments feeding new triples from the RDF stream into the C-SPARQL engine at two different rates: 5 triples per second (5 t/s) and 200 t/s. In both experiments, we measured the time required to compute the answer with the triples in the window.

It is possible to write a SPARQL query, which computes the same results as the C-SPARQL query above, by adding (a) a triple pattern that matches the creation date of the post, (b) a FILTER clause that selects the same time interval of the C-SPARQL query, and (c) an aggregate function that counts the number of topics using Jena SPARQL extensions for aggregates.

```
PREFIX sioc: <http://rdfs.org/sioc/ns#>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?topic count(?topic)
WHERE { ?post sioc:topic ?topic .
        ?post dcterms:created ?date .
        FILTER (?date > ''2009-07-20T22:47:00Z''^^xsd:dateTime
             && ?date < ''2009-07-20T22:50:00Z''^^xsd:dateTime ) }
GROUP BY ?topic
```

Using Jena, we executed the above SPARQL query six times against repositories containing the first 100, 500, 1000, 1500, 2000 and 2500 triples, respectively. In these experiments, we again measured the time required to compute each answer.

The results are shown in Figure 2 and are named *SPARQL*. Comparing the linear regressions of the three experiments, named *Linear(SPARQL)*, *Linear(C-SPARQL 5 t/s)* and *Linear(C-SPARQL 200 t/s)*, we see that the C-SPARQL window based selection always performs significantly better than the FILTER based selection of SPARQL in Jena. Notably, this result holds both for a low triple per second rate of 5 t/s and a reasonably high rate of 200 t/s.

6 Related Work

A previous effort to combine SPARQL and data streams is presented in Bolles et al. [26]. They introduce a syntax for the specification of logical and physical windows in SPARQL queries by means of local grammar extensions.

Our approach is different from their in several key aspects. First, Bolles et al. omit essential ingredients such as aggregate functions, thus the resulting expressive power is not sufficient to express interesting practical queries such as "Which are the top topics under discussion?". Second, the authors do not follow the approach, established by DSMS, to only use windows to transform streaming data into non-streaming data in order to apply standard algebraic operations. Bolles et al. choose to also change the standard SPARQL operators by making them timestamp-aware and, thereby, effectively introduce a new language semantics. Finally, their approach allows window clauses to appear within SPARQL

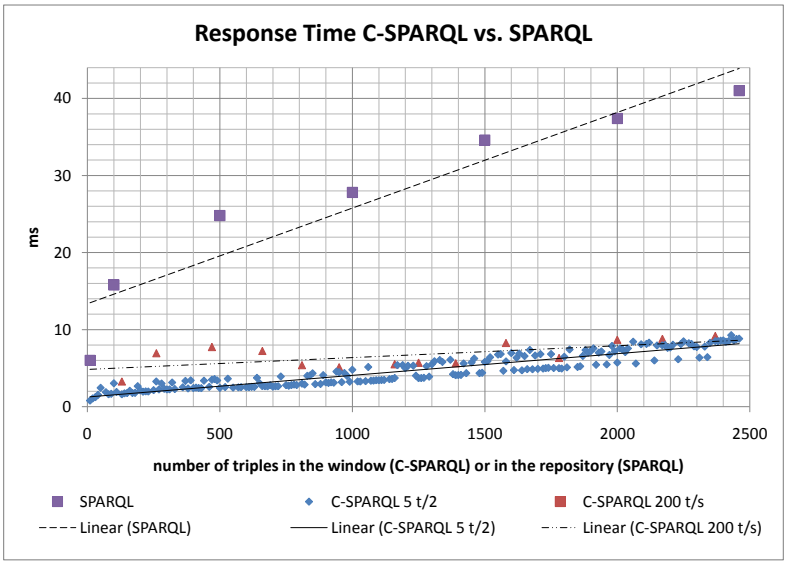

Fig. 2. The window based selection of C-SPARQL outperforms the FILTER based selection of SPARQL.

group graph pattern expressions. This makes the query syntax more intricate and it complicates query evaluation. Moreover, it violates the separation of concerns between stream management and query evaluation that is the basis of our simple architecture for C-SPARQL engines.

7 Conclusions

We began this paper with a list of questions related to social data on the Web that stress the streaming nature of blogs and microblogs. Our initial claim was that social data should not be treated as persistent data to be stored forever and queried on demand, but rather as transient data to be consumed on the fly by registered queries. In order to prove this claim, we have made the following arguments in this paper.

- RDF streams can be defined by extending RDF data type with a notion of timestamp;
- RDF streams can be serialized as N-Quads;
- sources of RDF streams are available and can be obtained from blogs and microblogs with the same approach used to obtain the RDF representations used for social semantic data;
- SPARQL can be extended with the notion of continuous query registered once and for all that keep monitoring such RDF streams, producing answers triggered by the streaming data and not by explicit invocation;

– C-SPARQL queries can be evaluated using a simple architecture based on the decision to keep stream management and query evaluation separated; and
– in terms of response time, even in a naive implementation of this architecture the window based selection of C-SPARQL outperforms the FILTER based selection needed to formulate the equivalent query in SPARQL.

Moreover, we do not exploit several optimization opportunities. On the one hand, we can adopt smarter query rewriting techniques that push part of a C-SPARQL query evaluation from the SPARQL engine to the DSMS. On the other hand, we are not considering the parallel nature of streams and thus the opportunity for parallel continuous query processing.

Finally, we have been limiting ourselves to treat RDF as relational data without considering it part of the Semantic Web stack. We are currently investigating techniques [27] to incrementally maintain materialization of knowledge derived by the triples currently selected by the window.

Acknowledgements

The work described in this paper has been partially supported by the European project LarKC (FP7-215535). Michael Grossniklaus's work is carried out under SNF grant number PBEZ2-121230.

References

1. Bojars, U., Breslin, J.G., Peristeras, V., Tummarello, G., Decker, S.: Interlinking the social web with semantics. IEEE Intelligent Systems **23**(3) (2008) 29–40
2. Passant, A., Hastrup, T., Bojars, U., Breslin, J.: Microblogging: A semantic web and distributed approach. In: 4th Workshop Scripting For the Semantic Web (SFSW2008) co-located with ESWC2008. (2008)
3. Nowack, B.: Semantic microblogging. In: Microblogging Conference. (2009)
4. Garofalakis, M., Gehrke, J., Rastogi, R.: Data Stream Management: Processing High-Speed Data Streams (Data-Centric Systems and Applications). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2007)
5. Della Valle, E., Ceri, S., Braga, D., Celino, I., Frensel, D., van Harmelen, F., Unel, G.: Research chapters in the area of stream reasoning. In: SR2009. Volume 466 of CEUR Workshop Proceedings., CEUR-WS.org (2009) online http://ceur-ws.org/Vol-466/sr2009-intro.pdf.
6. Golab, L., DeHaan, D., Demaine, E.D., López-Ortiz, A., Munro, J.I.: Identifying Frequent Items in Sliding Windows over On-line Packet Streams. In: Proc. Intl. Conf. on Internet Measurement (IMC 2003). (2003) 173–178
7. Jagadish, H.V., Mumick, I.S., Silberschatz, A.: View Maintenance Issues for the Chronicle Data Model. In: Proc. ACM Symp. on Principles of Database Systems (PODS 1995). (1995) 113–124
8. Liu, L., Pu, C., Tang, W.: Continual Queries for Internet Scale Event-Driven Information Delivery. IEEE Trans. Knowl. Data Eng. **11**(4) (1999) 610–628

9. Chen, J., DeWitt, D.J., Tian, F., Wang, Y.: NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In Chen, W., Naughton, J.F., Bernstein, P.A., eds.: Proc. ACM Intl. Conf. on Management of Data (SIGMOD 2000). (2000) 379–390
10. Balakrishnan, H., Balazinska, M., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Galvez, E., Salz, J., Stonebraker, M., Tatbul, N., Tibbetts, R., Zdonik, S.: Retrospective on Aurora. The VLDB Journal **13**(4) (2004) 370–383
11. Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Nishizawa, I., Rosenstein, J., Widom, J.: STREAM: The Stanford Stream Data Manager (Demonstration Description). In: Proc. ACM Intl. Conf. on Management of data (SIGMOD 2003). (2003) 665
12. Arasu, A., Babu, S., Widom, J.: The CQL Continuous Query Language: Semantic Foundations and Query Execution. The VLDB Journal **15**(2) (2006) 121–142
13. Babu, S., Widom, J.: Continuous Queries over Data Streams. SIGMOD Rec. **30**(3) (2001) 109–120
14. Law, Y.N., Wang, H., Zaniolo, C.: Query Languages and Data Models for Database Sequences and Data Streams. In: Proc. Intl. Conf. on Very Large Data Bases (VLDB 2004). (2004) 492–503
15. Law, Y.N., Zaniolo, C.: An Adaptive Nearest Neighbor Classification Algorithm for Data Streams. In: Proc. Europ. Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD 2005). (2005) 108–120
16. Bai, Y., Thakkar, H., Wang, H., Luo, C., Zaniolo, C.: A Data Stream Language and System Designed for Power and Extensibility. In: Proc. Intl. Conf. on Information and Knowledge Management (CIKM 2006). (2006) 337–346
17. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/
18. Gutierrez, C., Hurtado, C., Mendelzon, A.O.: Foundations of Semantic Web Databases. In: Proc. ACM Symp. on Principles of Database Systems (PODS 2004). (2004) 95–106
19. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and Complexity of SPARQL. In: Proc. Intl. Semantic Web Conf. (ISWC 2006). (2006) 30–43
20. Cyganiak, R.: A Relational Algebra for SPARQL. Technical report, HP-Labs
21. Haase, P., Broekstra, J., Eberhart, A., Volz, R.: A Comparison of RDF Query Languages. In: Proc. Intl. Semantic Web Conf. (ISWC 2004). (2004) 502–517
22. Carroll, J.J., Bizer, C., Hayes, P.J., Stickler, P.: Named graphs, provenance and trust. In: WWW. (2005) 613–622
23. Cyganiak, R., Harth, A., Hogan, A.: N-quads: Extending n-triples with context. http://sw.deri.org/2008/07/n-quads/ (2008)
24. Connolly, D., *et al.*: Gleaning Resource Descriptions from Dialects of Languages (GRDDL) - W3C Recommendation. Available on the Web at http://www.w3.org/TR/grddl/ (11 September 2007)
25. Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: C-SPARQL: SPARQL for continuous querying. In: WWW. (2009) 1061–1062
26. Bolles, A., Grawunder, M., Jacobi, J.: Streaming SPARQL – Extending SPARQL to Process Data Streams. In: Proc. Europ. Semantic Web Conf. (ESWC 2008). (2008) 448–462
27. Volz, R., Staab, S., Motik, B.: Incrementally maintaining materializations of ontologies stored in logic databases. J. Data Semantics **2** (2005) 1–34