SRBench: A Streaming RDF/SPARQL Benchmark

Ying Zhang[1], Pham Minh Duc[1], Oscar Corcho[2], and Jean-Paul Calbimonte[2]

[1] Centrum Wiskunde & Informatica, Amsterdam, The Netherlands
{Y.Zhang, P.Minh.Duc}@cwi.nl
[2] Universidad Politécnica de Madrid, Spain
ocorcho@fi.upm.es, jp.calbimonte@upm.es

Abstract. We introduce *SRBench*, a general-purpose benchmark primarily designed for streaming RDF/SPARQL engines, completely based on real-world data sets from the Linked Open Data cloud. With the increasing problem of too much streaming data but not enough tools to gain knowledge from them, researchers have set out for solutions in which Semantic Web technologies are adapted and extended for publishing, sharing, analysing and understanding streaming data. To help researchers and users comparing streaming RDF/SPARQL (strRS) engines in a standardised application scenario, we have designed SRBench, with which one can assess the abilities of a strRS engine to cope with a broad range of use cases typically encountered in real-world scenarios. The data sets used in the benchmark have been carefully chosen, such that they represent a realistic and relevant usage of streaming data. The benchmark defines a concise, yet comprehensive set of queries that cover the major aspects of strRS processing. Finally, our work is complemented with a functional evaluation on three representative strRS engines: $\text{SPARQL}_{\text{Stream}}$, C-SPARQL and CQELS. The presented results are meant to give a first baseline and illustrate the state-of-the-art.

1 Introduction

Unlike the static data, which are known *a priori* and rarely change, streaming data arrive as continuous streams typically at high rates, e.g., once per second or even higher. For data streams, the most recent data are usually most relevant, and the queries mainly focus on the continuous changes of the observed properties over time. The amount of streaming data has been growing extremely fast in the past years and is expected to grow even faster in the coming decades. However, existing Data Stream Management Systems (DSMSs) are not able to capture *all* information from the available streaming data, letting alone interlinking those data with other data sets to derive implicit information [5, 16]. In the meantime, Semantic Web techniques have focused on how to publish and interlink data on the World Wide Web, and how to perform complex reasoning tasks on the data. However, these techniques have generally not taken into account rapidly changing streaming data. The lack of integration and communication between different streaming data resources often isolates important data streams and intensifies the existing problem of "too much (streaming) data but not enough (tools to gain and derive) knowledge" [32]. To tackle this problem, researchers have set out for solutions in which Semantic Web techniques are adapted and extended for publishing, sharing, analysing and understanding of streaming data.

Sheth et al. [32] first envisioned a Semantic Sensor Web (SSW), in which sensor data are annotated with semantic metadata to increase interoperability and provide contextual information essential for situational knowledge[3]. Subsequently, Corcho et al. [14] identified the five most relevant challenges of the current SSW. Della Valle et al. [16] proposed a novel approach, called *stream reasoning*, to provide the abstractions, foundations, methods and tools required to integrate data streams, the Semantic Web and reasoning systems. Sequeda et al. [31] introduced the concept of Linked Stream Data (LSD) which applies the Linked Data principles to streaming data, so that data streams can be published as part of the Web of Linked Data. So far, these visions have been answered by various proposals to address the topic of *streaming data processing using Semantic Web technologies* from different angles. For instance, how to apply reasoning on streaming data [1, 16, 32, 35, 36]; how to publish raw streaming data and connect them to the existing data sets on the Semantic Web [10, 14, 26, 31, 32]; and how to extend the SPARQL query language to process streaming data [6, 9, 11, 20, 24, 25]. The increasing interest in streaming RDF/SPARQL (strRS) engines calls for a *standard way* to compare the functionality and performance of different systems.

So far, little work has been done on benchmarking DSMSs. The Linear Road benchmark [3] is the only publicly available DSMSs benchmark. However, it is not ideal when used to assess strRS engines. As originally designed to evaluate traditional DSMSs, the benchmark is based on the relational data model, so it does not capture the properties of RDF graph data. Moreover, Linear Road does not consider interlinking the benchmark data set with other data sets; neither does it address reasoning. In Semantic Web, existing RDF/SPARQL benchmarks, e.g., [8, 21, 30], have been focused on static data, so they do not capture the aforementioned dynamic properties of streaming data. In [24, 25], some microbenchmark queries are used for preliminary evaluations of the proposed strRS systems. However, the queries were created with a particular system in mind and they only cover a small subset of the features of SPARQL. Hence, they cannot serve as general-purpose benchmarks.

In this paper, we present ***SRBench***, a streaming RDF/SPARQL benchmark that aims at assessing the abilities of strRS engines in dealing with important features from both DSMSs and Semantic Web research areas combined in one real-world application scenario. That is, how well can a system cope with a broad range of different query types in which Semantic Web technologies, including querying, interlinking, sharing and reasoning, are applied on highly dynamic streaming RDF data. The benchmark can help both researchers and users to compare strRS engines in a pervasive application scenario in our daily life, i.e., querying and deriving information from weather stations. To the best of our knowledge, SRBench is the first *general-purpose* benchmark that is *primarily* designed to compare strRS engines.

Given the importance of interlinked data sets in Semantic Web, and the study of Duan et al. [17], which points out that the synthetic data used by the existing RDF benchmarks generally do not accurately predict the behaviour of RDF stores in realistic scenarios, we decided to use a real-world sensor data set, i.e., LinkedSensorData [27], from the Linked Open Data (LOD) cloud [34] as the basic data set of SRBench. To

[3] *Situational knowledge* is the knowledge specific to a particular situation.

assess a system's ability of dealing with interlinked data, we additionally use the LOD data sets GeoNames [18] and DBpedia [15], which are linked to the LinkedSensorData.

SRBench defines a concise, yet comprehensive set of queries which covers the major aspects of strRS query processing, ranging from simple graph pattern matching queries only on streaming data to queries requiring reasoning over multiple interlinked data sets. Each query is intended to challenge a particular aspect of the query processor. The main advantages of applying Semantic Web technologies on streaming data include providing better search facilities by adding semantics to the data, reasoning through ontologies, and integration with other data sets. The ability of a strRS engine to process these distinctive features is accessed by the benchmark with queries that apply reasoning not only over the streaming sensor data, but also over the sensor metadata and the two aforementioned LOD data sets.

Given that existing strRS engines are still in their infancy, we deem it important to first conduct a functional evaluation. Do they provide a sufficient set of functions that are required by the streaming applications? Do they miss any crucial functionalities? Do they provide any additional functionalities that can be beneficial for streaming applications, which thus distinguish themselves from similar systems? Therefore, we complement our work on SRBench by a functional evaluation on three strRS engines, $\text{SPARQL}_{\text{Stream}}$ [11], C-SPARQL [6] and CQELS [25]. Each of these systems also proposes its own SPARQL extension for streaming data processing. The evaluation is not meant to be an exhaustive examination of all existing strRS systems. The testing systems are chosen, because they represent different approaches in strRS processing. $\text{SPARQL}_{\text{Stream}}$ aims at enabling ontology-based access to streaming data. C-SPARQL attempts to facilitate reasoning upon rapidly changing information. CQELS is the only native strRS system built from scratch. The evaluation results are intended to give a first baseline and illustrate the state-of-the-art.

The target audience of this paper can be divided into three groups. First, the framework presented here can help strRS engine implementers to verify and refine their query processors by comparing them to other implementations. Second, users can be assisted in choosing between products by using SRBench as a simple case study or pilot project that yet provides essential ingredients of the targeted system. For researchers, lastly, we provide a framework for helping to tailor existing technologies for use in streaming settings and for refinement or design of algorithms.

This paper is further organised as follows. Section 2 discusses design challenges of the benchmark. Section 3 describes the benchmark data sets. Section 4 defines the benchmark queries. Section 5 presents the results of the functional evaluation. Finally, we discuss related work in Section 6 and conclude in Section 7.

2 Design Challenges

In this section, we discuss the unique challenges that streaming RDF/SPARQL processing imposes on the design of a benchmark and how they are met by SRBench.

Proper Benchmark Data set First of all, the design of a streaming RDF/SPARQL benchmark requires a cautiously chosen data set that is relevant [19], realistic [17], semantically valid [3] and interlinkable [31]. Additionally, the data set should allow the

formulation of queries that both feel natural and present a concise but complete set of challenges that strRS engines should meet.

In SRBench, this challenge is met by choosing the LinkedSensorData [27] data set from the LOD cloud as the basic data set. Among different kinds of streaming data[4], sensor data is a major class of streaming data with the longest history. Weather information applications have long become pervasive in our daily life, in addition, they are gaining increasing social and financial values in more accurate prediction of extreme weather conditions. The LinkedSensorData is a real-world data set containing the US weather data published by Kno.e.sis[5] according to the LSD principles [13], which applies the well-established Linked Data principles [7] to streaming data. The LinkedSensorData is the first and, so far, largest LSD data set in the LOD cloud [34] and CKAN [12] containing ~1.7 billion triples.

To assess a system's ability of dealing with interlinked data, we additionally use other data sets from the LOD cloud. Currently, this includes the GeoNames [18] and DBpedia [15] data sets. Our choice for the GeoNames data set is determined by the fact that the LinkedSensorData data set links the sensor locations to nearby geographic places defined by the GeoNames data set. The choice for the DBpedia data set is a matter of course, since DBpedia is the largest and most popularly used data set in the LOD cloud. By using the LOD data sets, it is easy to extend the benchmark with more data sets in the future. This enables adding more semantics to the benchmark's application scenario, which subsequently allows more use cases.

A Concise Set of Features The main advantages of applying Semantic Web technologies on streaming data include providing better search and sharing facilities by adding semantics to the data, reasoning through ontologies, and integration with other data sets. The benchmark should provide a comprehensive set of queries that assess a system's ability of processing these distinctive features on highly dynamic (in terms of arriving rate and amount) streaming data, possibly in combination with static data. The queries should have different levels of complexity, so that the benchmark can be used to evaluate not only general purpose systems supporting a broad spectrum of features, but also specialised systems aiming at providing a limited number of features with high efficiency. Nonetheless, as stated by the "20 queries" principles [23], the number of queries should be compact. Thus, the number and types of queries should exhibit a good balance between conciseness and detail making it possible to run the benchmark in an acceptable time, while still acquiring interesting characteristics of the system(s) tested.

In SRBench, this challenge is met by a set of seventeen queries that have been carefully chosen such that they provide valuable insights that can be generally applied to strRS systems and are useful in many domains, e.g., notion of time bounded queries (e.g., data in the latest X units-of-time); notion of continuous queries (i.e., queries evaluated periodically); data summarisation in the queries (e.g., aggregates); providing high-level information from raw-data (e.g., ask for hurricanes, while the raw-data are simply temperature, wind measurements); and combining streams with contextual static data. Before designing the queries, we first identified the set of important features in the SPARQL 1.1 language [22] and streaming data processing. Then, use cases are

[4] Next to sensor data streams, there are text streams and video streams.

[5] `http://knoesis.wright.edu`

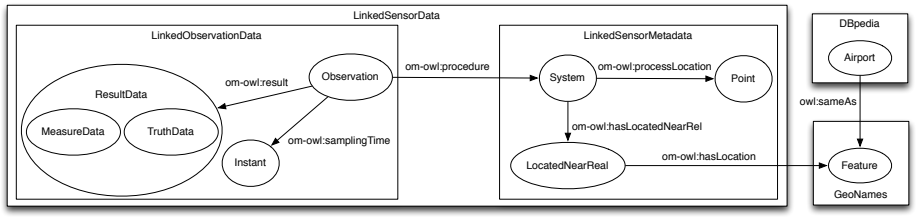

Fig. 1. An overview of the data sets used in SRBench and their relationships.

carefully chosen such that they reflect how the weather information is used in the real world, while each of them challenges the query processor, with focus on one or two of the important features (but not limited to).

No Standard Query Language A standard query language for streaming data processing has never come into existence. Therefore, the queries of a streaming benchmark should be specified in a language agnostic way, yet have a clear semantics.

In SRBench, this challenge is met by first giving a descriptive definition of the benchmark queries, in a similar way as how the Berlin SPARQL Benchmark describes its queries[6]. Then, we provide implementations of the benchmark queries using the three major SPARQL extensions for streaming data processing (for short: *streaming SPARQL*), i.e., $\text{SPARQL}_{\text{Stream}}$, C-SPARQL and CQELS. Thus, these three sets of implementing queries are not only used by the functional evaluation in Section 5, but also for the purpose of clarifying the benchmark query definitions.

3 Data sets

In this section, we briefly describe the three LOD data sets used by SRBench. An overview of the data sets and their ontologies, and how they are linked to each other is shown in Figure 1. More information of the data sets can be found in [37].

The LinkedSensorData Data Set Work on producing Linked Data from data emitted by sensors was initiated in 2009, pioneered by [31, 26]. The LinkedSensorData contains the US weather data collected since 2002 by MesoWest[7], and were transformed into LSD by Kno.e.sis. LinkedSensorData contains two sub-datasets. The *LinkedSensorMetadata* contains expressive descriptions of ~20,000 weather stations in the US. On average, there are five sensors per weather station, so there are in total ~100,000 sensors in the data set. The sensors measure phenomena such as temperature, visibility, precipitation, pressure, wind speed and humidity. In addition to location attributes, e.g., latitude, longitude, and elevation, there are also links to locations in GeoNames that are near the weather stations. The *LinkedObservationData* contains expressive descriptions of hurricane and blizzard observations in the US. The observations collected include values of all phenomena measured by the sensors. The data set includes observations within the entire US during the time periods that several major storms were

[6] `http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/`
[7] `http://mesowest.utah.edu/`

Name	Storm Type	Date	#Triples	#Observations	Data size
ALL			1,730,284,735	159,460,500	~111 GB
Bill	Hurricane	Aug. 17 – 22, 2009	231,021,108	21,272,790	~15 GB
Ike	Hurricane	Sep. 01 – 13, 2008	374,094,660	34,430,964	~34 GB
Gustav	Hurricane	Aug. 25 – 31, 2008	258,378,511	23,792,818	~17 GB
Bertha	Hurricane	Jul. 06 – 17, 2008	278,235,734	25,762,568	~13 GB
Wilma	Hurricane	Oct. 17 – 23, 2005	171,854,686	15,797,852	~10 GB
Katrina	Hurricane	Aug. 23 – 30, 2005	203,386,049	18,832,041	~12 GB
Charley	Hurricane	Aug. 09 – 15, 2004	101,956,760	9,333,676	~7 GB
	Blizzard	Apr. 01 – 06, 2003	111,357,227	10,237,791	~2 GB

Table 1. Statistics of the LinkedObservationData data sets used by SRBench

active, including Hurricane Katrina, Ike, Bill, Bertha, Wilma, Charley, Gustav, and a major blizzard in Nevada in 2003. These observations are generated by weather stations described in the LinkedSensorMetadata data set introduced above. Currently, this data set contains almost two billion RDF triples, which together describe more than 159 million observations. For SRBench, we have obtained all linked sensor observation data sets from the original Kno.e.sis site for LinkedSensorData [27]. Table 1 shows the statistics of the LinkedObservationData data sets as presented on the original website, to which we have added the sizes of the data sets after they have been unpacked.

All data are described according to the *sensor-observation ontology* [27]. The ontology class `System` describes a weather sensor station, e.g., its ID and location of the station, a geographical location to which the station is located nearby and the weather properties observed by this station. The class `Observation` describes an observation made by a weather sensor station, e.g., the ID of the weather station that has made the observation, the type of the observed weather property and the value and time of the observation. The class `MeasureData` describes the numerical value of an observation, while the class `TruthData` describes the truth-value of an observation. The class `LocatedNearRel` describes a geographic location to which a weather sensor station is located nearby, e.g., the distance between the nearby location and the sensor station. The class `Point` describes a geographic point location, in terms of latitude, longitude and altitude. The class `Instant` describes a date/time object.

The GeoNames Data Set is a free geographical database that covers all countries and contains >8 million place names [18]. For SRBench, we use version 3.01 of the *GeoNames ontology* [18]. Its main class is `Feature`, which describes a geographical location, e.g., its names, its latitude and longitude, the country to which this location belong and other locations that are close to this location. We have obtained the dump of the complete GeoNames RDF data set, which contains ~8 million geographic features with ~46 million RDF triples. The dump has one RDF document per toponym. The complete data set occupies ~10GB on disk.

The DBpedia Data Set The DBpedia ontology is highly complex but well documented, so we do not repeat its class definitions here, but refer the interested readers to its official website [15] instead. For SRBench, we have obtained the data sets from the English language collection, which consists of 44 RDF files in N-triple format with ~181 million triples. The total data size is ~27 Gigabytes. The DBpedia data set is directly linked to the GeoNames data set through the `owl:sameAs` property. DBpedia has in total 85,000 links to the GeoNames data set.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17
1 graph pattern matching	A	A,F,O	A	A,F	A	A,F,U	A	A	A	A	A,F	A,F,U	A,F	A,F,U	A,F	A,F	A,F
2 solution modifier	P,D	P,D	P	P	P	P	P,D	P	P	P,D	P,D	P	P	P,D	P	P	P
3 query form	S	S	A	S	C	S	S	S	S	S	S	S	S	S	S	S	S
4 SPARQL 1.1		F,P	A	A,E,M,F	A,S		N	A,E,M	A,E,M		A,S,M,F	A,S,E,M,F,P	A,E,M,F,P	F,P	A,E,M,P	P	P
5 reasoning			R												C	A	C
6 streaming feature	T	T	T	T	T	T	T,	T	T	T	T	T	T	T	T		
7 data access	O	O	O	O	O	O	O	O, S	O,S	O,S	O,S	O,S,F	O,S,G	O,S,G	O,S,D	O,S,G,D	S

Table 2. Addressed features per query. Operators are abbreviated in per row unique capital letters, defined as: 1. **A**nd, **F**ilter, **U**nion, **O**ptional; 2. **P**rojection, **D**istinct; 3. **S**elect, **C**onstruct, **A**sk; 4. **A**ggregate, **S**ubquery, **N**egation, **E**xpr in SELECT, assign**M**ent, **F**unctions&operators, **P**roperty path; 5. sub**C**lassOf, subp**R**opertyOf, owl:same**A**s; 6. **T**ime-based window, **I**stream, **D**stream, **R**stream; 7. Linked**O**bservationData, Linked**S**ensorMetadata, **G**eoNames, **D**bpedia.

4 Benchmark Queries

In this section, we define the SRBench benchmark queries. An overview of the language features addressed by each query is given in Table 2.

A SPARQL query can be divided into three parts. As SPARQL is essentially a graph-matching query language, *graph pattern matching* is the fundamental and one of the most complex parts of a SPARQL query. This part includes features such as the basic graph pattern matching operators '.' (representing a natural join `AND`) and `FILTER`, and the most complicated operators `UNION` and `OPTIONAL` [28]. All graph pattern matching operators are addressed in the SRBench queries.

The second part is called the *solution modifiers*, which are used to modify the results of the graph pattern matching operators. The solution modifiers contain six operators, i.e., projection, `DISTINCT`, `ORDER BY`, `OFFSET`, `LIMIT` and `REDUCED`. In the SRBench queries, only the projection and `DISTINCT` solution modifiers are addressed, because the additional values of the other four operators are negligible in streaming applications. `ORDER BY` is ignored since streaming data are already sorted by their time stamps, and sorting the results on another attribute will only produce partially sorted data (within one window). The features of `OFFSET` and `LIMIT` are largely covered by sliding windows, which are more appropriate for strRS queries. Finally, the nondeterministic property of `REDUCED` highly complicates the verification of the query results.

The last part is called the *query forms*, which determine the form of the final output of a SPARQL query. The output can be one of the four query forms: `SELECT` returns the projected variables bound in a query pattern match; `CONSTRUCT` returns a new RDF graph constructed by substituting variables in a set of triple templates; `ASK` returns a boolean indicating whether a query pattern matches or not; and `DESCRIBE` returns an RDF graph that describes the resources found. In the SRBench queries, the `DESCRIBE` form is not used, because the result of a `DESCRIBE` query is highly implementation dependant, which largely complicates the verification of the query results. Moreover, the functionality of `DESCRIBE` can be approximated using explicit graph pattern matching and projections. The first three rows of Table 2 respectively survey how the operators of the three parts are distributed among the benchmark queries.

Next to the features defined by SPARQL 1.0 [29], SPARQL 1.1 has introduced several new features, including aggregates, subqueries, negation, expressions in the `SELECT`

clause, Property Paths, assignment, a short form for `CONSTRUCT`, and an expanded set of functions and operators. Except the short form of `CONSTRUCT`, which is merely a syntax sugar, we make extensive use of these new features in the benchmark queries, especially the Property Paths expressions, which we regard as a major contribution of SPARQL 1.1 that provides great flexibility to navigate through the RDF graphs. Note that, since SPARQL 1.1 is still a W3C working draft, changes to the syntax and/or semantics of the new language features are possible. For instance, the semantics of the Property Path expressions might be modified, due to recent analysis of the unfeasibility of their current semantics [4]. Possible changes in SPARQL 1.1 will not affect the definition of the benchmark queries, since they are specified independent of any query language. Row 4 of Table 2 surveys how the SPARQL 1.1 new features are distributed among the benchmark queries.

A main added value of applying Semantic Web technologies on streaming data is the possibility of reasoning over the data, so SRBench includes queries that allow exploiting such facility if provided by the processing engine. Currently, the queries involve reasoning over the `rdfs:subClassOf`, `rdfs:subPropertyOf` and `owl:sameAs` properties. Note that, these queries can be implemented and executed by both systems with and without inference mechanisms, but the differences might be noticeable in the query results. That is, systems with inference mechanisms will most probably return more results than systems without such mechanisms. Also note that, although SPARQL is not a reasoning language, it can be used to query ontologies if they are encoded in RDF. So, on systems without reasoning, this shortcoming can be alleviated by explicitly expressing reasoning tasks using extra graph patterns with Property Path over the ontologies. In our functional evaluation (Section 5), we actually use this workaround to implement the benchmark queries using the three language extensions tested. Row 5 of Table 2 surveys how reasoning features are distributed among the benchmark queries.

Although there is no standard query language for streaming RDF data, existing streaming SPARQL extensions generally introduce streaming data operators that are inspired by the continuous query language CQL [2]. So, next to the classical SPARQL 1.0 and 1.1 operators, we add three important streaming SPARQL features. The *time-based sliding window* operator is a basic operator in streaming data processing, which allows users to control data access using time intervals. A slide size can be defined to create overlapping or disjoint windows. The *window-to-stream* operators, *Istream* and *Dstream*, return data items that have been inserted or deleted since the previous window, respectively. Although not frequently used, these operators help to detect changes in the data streams, a feature particularly important for streaming data. Row 6 of Table 2 surveys how the streaming operators are distributed among the benchmark queries.

To conclude, the SRBench queries are designed based on a real use case in LSD. They cover the most important SPARQL operators and the common streaming SPARQL extensions. SRBench clearly shows the added values of the Semantic Web technologies on gaining and even deriving knowledge from streaming data. The benchmark provides a general framework to assess the ability of streaming RDF/SPARQL engines to support such applications. In the reminder of this section, the queries are grouped under section headings which indicate the features to be tested. Due to lack of space, we omit presenting the query implementations, which are available at [33].

4.1 Basic Pattern Matching

Q1. Get the rainfall observed once in an hour.

This is a basic but important query. It tests an engines ability to handle basic graph patterns, disjoint time windows ("once in an hour") to gain knowledge about the mostly spoken topic ("rainfall"), when talking about the weather.

4.2 Optional Pattern Matching

Q2. Get all precipitation observed once in an hour.

Although similar to Q1, this query is much more complex, because it requires returning all types of precipitation. Since the triple patterns for different kinds of precipitations maybe different, `OPTIONAL` patterns are needed to capture the possible differences. Additionally, this query exploits an engine's ability of reasoning over all instances of the class `PrecipitationObservation` and its subclasses.

4.3 ASK Query Form

Q3. Detect if a hurricane is being observed.

A hurricane has a sustained wind (for >3 hours) of at least 74 miles per hour. This query continuously monitors if the weather conditions observed in the current time window (i.e., an hour) is extreme ("a hurricane"). It also tests the engines ability to filter out the minimal amount of the streaming data to quickly compute the answer.

4.4 Overlapping Sliding Window and Historical Data

Q4. Get the average wind speed at the stations where the air temperature is >32 degrees in the last hour, every 10 minutes.

Combine values observed for multiple weather properties. This query tests the engines ability to deal with historical data that need to be (temporarily) stored. Moreover, contrary to queries for which an incoming data item can be immediately consumed and then discarded, this query tests how efficient an engine's strategy is to decide how to store historical data and for how long.

4.5 CONSTRUCT Derived Knowledge

Q5. Detect if a station is observing a blizzard.

A blizzard is a severe snow storm characterised by low temperatures, strong winds and heavy snow lasting for at least three hours. This query detects extreme weather conditions by combining multiple observed weather properties. It tests the engines ability to produce new knowledge derived by combining existing data.

4.6 Union

Q6. Get the stations that have observed extremely low visibility in the last hour.

Next to direct measurements of low visibility (<10 centimetres), heavy snowfall and rainfall (> 30 centimetres) also cause low visibility. This is a more complex example of detecting extreme weather conditions, which requires not only gaining knowledge explicitly contained in the data (i.e., visibility), but also deriving implicit knowledge from data sources (i.e., snowfall and rainfall).

4.7 Window-to-Stream operation

Q7. Detect stations that are recently broken.

If a station suddenly stops producing (observation) data, it might be broken. Knowing the stability of the stations is an important issue, which can be deduced from absent

data. This query tests the engines ability to cope with the dynamic properties that are specific for streaming data.

4.8 Aggregates

Q8. Get the daily minimal and maximal air temperature observed by the sensor at a given location.

Temperature is the most common weather condition queried. This query tests the engines' ability to aggregate data grouped by their geo-spatial properties.

4.9 Expression in SELECT Clause

Q9. Get the daily average wind force and direction observed by the sensor at a given location.

Wind is the other most commonly queried weather condition. The Beaufort Wind Force Scale[8] is an international standard to express how strong the wind is. It attaches some semantics to the bare wind speed numbers. Since this query requires wind speeds to be converted into Beaufort scales, it tests the engines ability to post process the qualified triple patterns.

4.10 Join with Static Data

Q10. Get the locations where a heavy snowfall has been observed in the last day.

This query finds places that are suitable for a ski holiday. It also tests the engines ability to join the dynamic sensor streaming data with the static sensor metadata.

4.11 Subquery

Q11. Detecting if a station is producing significantly different observation values than its neighbouring stations.

Detecting malfunctioning sensors is an important issue in all sensor systems. If two sensor stations are located close (denoted by `hasLocatedNearRel`) to the same location, the sensors are neighbours of each other and they should observe similar weather conditions, otherwise, a sensor might be malfunctioning. This query tests the engines ability to compute complex subqueries.

4.12 Property Path Expressions

This group of queries tests the engines ability to derive knowledge from multiple interlinked data sets using Property Path expressions. In particular, the queries require computing paths with arbitrary lengths for the `parentFeature` relationship, and computing alternatives for the name of the resulting places.

Q12. Get the hourly average air temperature and humidity of large cities.

To analyse air pollution in large cities, one might want to know if the temperature is higher during the rush hours in such cities. This query requires using the GeoNames data set to find large cities, i.e., population > 15000, and use the `hasLocatedNearRel` property in the sensor ontology [27] to find sensors located in or near to these cities.

Q13. Get the shores in Florida, US where a strong wind, i.e., the wind force is between 6 and 9, has been observed in the last hour.

This query finds shores in Florida, US, where one can go windsurfing now. It requires first reasoning over the `parentADM{1,2,3,4}` and `parentFeature` properties of the

[8] `http://en.wikipedia.org/wiki/Beaufort_scale`

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17
$SPARQL_{Stream}$	✓	PP	A	G	G	✓	✓	G	G,IF	SD	SD	PP,SD	PP,SD	PP,SD	PP,SD	PP,SD	PP,SD
CQELS	✓	PP	A	✓	✓	✓	D/N	✓	IF	✓	✓	PP	PP	PP	PP	PP	PP
C-SPARQL	✓	PP	A	✓	✓	✓	D	✓	IF	✓	✓	PP	PP	PP	PP	PP	PP

Table 3. Results of the functional evaluation. The ticks indicate queries supported by an engine. Uppercase letters are abbreviations of features required by a query that are not supported by a particular system. The abbreviations are defined as the following: **A**sk; **D**stream; **G**roup by and aggregations; **IF** expression; **N**egation; **P**roperty **P**ath; and **S**tatic **D**ataset. The '/' symbol means 'or', i.e., the query would work if one of the listed features is available. The ',' symbol means 'and', i.e., all listed features are needed by the query.

GeoNames ontology to find the shores in Florida, US; and then using the `hasLocatedNearRel` property in the sensor ontology to find sensors located near to these shores.

Q14. Get the airport(s) located in the same city as the sensor that has observed extremely low visibility in the last hour.

This query triggers an alarm if a dangerous weather condition has been observed. It requires using the GeoNames data set and the `hasLocatedNearRel` property in the sensor ontology to find airport(s) and sensors located in the same city.

4.13 Ontology-based Reasoning

This group of queries exploit the engines ability to apply reasoning, using the properties `rdfs:subClassOf` and `owl:sameAs`, over the ontologies of the interlinked data sets.

Q15. Get the locations where the wind speed in the last hour is higher than a known hurricane.

By comparing an observed value with historical values, we can detect extreme weather conditions. This query requires reasoning over `rdfs:subClassOf` to find all known hurricanes in the system.

Q16. Get the heritage sites that are threatened by a hurricane.

We want to trigger an alarm if a dangerous weather condition has been observed. This query requires using a Property Path expression with an arbitrary length path to find all heritages sites in the DBpedia data set; then reasoning using `owl:sameAs` to link a heritage to a geographical instance described by the GeoNames data; and finally using the GeoNames data set and the `hasLocatedNearRel` property in the sensor ontology to find sensors located close to the monuments.

Q17. Estimate the damage where a hurricane has been observed.

A first thing we want to know after a natural disaster is the damage it brings. This can be estimated by consulting the damage brought by similar disasters that have happened before in the same/nearby area. This query requires using the DBpedia data set to find the damages caused by earlier hurricanes in the same/nearby area as the sensor that has observed a hurricane.

5 Implementation and Evaluation

As streaming RDF/SPARQL processing is a new topic with less than a decade of history, the proposed systems are mostly in a beginning stage of development. During

this phase, one of the most important issues is to assess the effectiveness of the proposed systems. We have complemented our work on SRBench with a functional evaluation on three leading strRS systems, i.e., SPARQL$_{\text{Stream}}$ [11], CQELS [25] and C-SPARQL [6], that address strRS processing from different angles. In the evaluation, we seek answers to the following questions: Do they provide an adequate set of functionalities that are needed by the streaming applications? Do they sufficiently reveal the additional value of RDF/SPARQL for streaming data processing? Furthermore, do they provide any additional interesting functionalities, which help the users to distinguish one system from the others? The results presented in this section are meant to give a first insight into the state-of-the-art of the strRS engines and highlight several pros and cons of different approaches.

All three systems come with a streaming SPARQL language proposal and a query processing engine. The queries are implemented according to the syntax specified in [11], [25] and [6] for SPARQL$_{\text{Stream}}$, CQELS and and C-SPARQL, respectively. To execute the queries, we downloaded the latest SPARQL$_{\text{Stream}}$[9], CQELS (Aug. 2011)[10] and C-SPARQL 0.7.4[11]. All implementing queries can be found in the SRBench wiki page [33]. An overview of the evaluation results is shown in Table 3. A tick indicates that the engine is able to process a particular query. For each query that cannot be processed by a certain engine, we denote the main missing feature(s) that cause the query to fail. The abbreviations are defined in the table caption.

The results of the evaluation are fairly close to our expectation. In general, all three engines support basic SPARQL features (i.e., the graph pattern matching features, solution modifiers and the `SELECT` and `CONSTRUCT` query forms discussed in Section 4) over time-based windows of streaming data. The main issue we have identified from the evaluation is that all three engines' abilities of dealing with the new features introduced by SPARQL 1.1 are rather limited. So, for instance, seven out of seventeen queries cannot be executed, because the Property Path expressions are not supported by any of the tested engines. We regard the Property Path expressions as one of the most important features of SPARQL 1.1, because it provides flexible ways to navigate through RDF graphs and facilitates reasoning over various graph patterns. This feature does not exist in other query languages, e.g., SQL and XQuery, that are not primarily designed to query graph data. The lacking of support for the SPARQL 1.1 features is most probably due to the freshness of the SPARQL 1.1 language. However, we would like to emphasise that the advanced SPARQL 1.1 features are the distinguishing factors of the usefulness of one system from others for streaming RDF applications.

Several more remarks can be made from Table 3. The lack of support for the `ASK` query form needed by Q3 is unexpected, since it is an easy to implement feature. This might be caused by that it does not have research values, but in real-world scenario's, such as the one used by SRBench, the users often start with asking for basic information that can be just as simple as "is it raining?". Q3 can also be implemented using the `IF` function of SPARQL 1.1, but it is available on none of the tested engines.

[9] `http://code.google.com/p/semanticstreams/source/checkout`
[10] `http://code.google.com/p/cqels/`
[11] `http://streamreasoning.org/download`

Q7 can be most easily implemented using the window-to-stream operator `DSTREAM`, which can detect data items that have been deleted since the previous window, which is exactly what this query asks. Although all three systems provide ways to produce new data streams from the results of a continuous query, $\text{SPARQL}_{\text{Stream}}$ is the only streaming SPARQL extension that support `DSTREAM`. Q7 can also be implemented by querying the same stream twice with different window definition and then using the `NOT EXISTS` expression of SPARQL 1.1 to test the absence of a graph pattern from the previous time interval in the current time interval. Since CQELS allows defining different time windows for the same stream, this query can be expressed in the CQELS language, but query execution failed because the CQELS engine does not support `NOT EXISTS` yet. C-SPARQL does not allow query the same stream more than once, and SPARQL 1.1 does not define arithmetic functions over the date/time data type, it is not possible to express the query in C-SPARQL.

The number of functionalities currently supported by $\text{SPARQL}_{\text{Stream}}$ is somewhat less than those supported by CQELS and C-SPARQL. As the `GROUP BY` and aggregations functions are still work in progress in $\text{SPARQL}_{\text{Stream}}$, it causes four more queries (i.e., Q4, Q5, Q8 and Q9) to fail. Moreover, the development of the $\text{SPARQL}_{\text{Stream}}$ engine has so far concentrated on supporting streaming data, but not both streaming and static data. This is one of the main reasons that the queries Q10 – 17 cannot be run on $\text{SPARQL}_{\text{Stream}}$. Enabling queries on both streaming and static data sets is an ongoing subproject of PlanetData[12].

Little work has been done on enabling reasoning over streaming data. C-SPARQL is the only testing system that supports reasoning based on simple RDF entailment. $\text{SPARQL}_{\text{Stream}}$ and CQELS currently do not tackle the problem of reasoning, because $\text{SPARQL}_{\text{Stream}}$ targets at enabling ontology based access to streaming data, while CQELS concentrates on building a strRS engine from scratch. So, next to the Property Path expressions, the ability to apply reasoning is another distinguishing factor.

The overall conclusion of our evaluation is that there is no single best system yet, even thought the $\text{SPARQL}_{\text{Stream}}$ engine supports fewer queries than CQELS and C-SPARQL. Both the SPARQL 1.1 language and the streaming RDF/SPARQL engines have been introduced only recently. As the time passes, we expect the strRS engines to gradually provide a richer set of functionalities.

Although this work focuses on a functional evaluation, we propose a number of metrics that should be used for a performance evaluation using SRBench. *Correctness*: the query results must be validated, taking into account possible variations in ordering, and possibly multiple valid results per query. The validation results should be expressed in terms of *precision* and *recall*. *Throughput*: the maximal number of incoming data items a strRS engine is able to process per time unit. *Scalability*: how does the system reacts to increasing number of incoming streams and continuous queries to be processed. *Response time*: the minimal elapsed time between a data item entering the system and being returned as output of a query. Note that response time is mainly relevant for queries allowing immediate query results upon receiving of a data item.

[12] `http://planet-data-wiki.sti2.at/`

6 Related Work

The Linear Road Benchmark [3] is the only publicly available benchmark developed for evaluating traditional data stream engines. The benchmark simulates a traffic management scenario where multiple cars are moving on multiple lanes and on multiple different roads. The system to be tested is responsible to monitor the position of each car, and continuously calculates and reports to each car the tolls it needs to pay and whether there is an accident that might affect it. In addition, the system needs to continuously maintain historical data, as it is accumulated, and report to each car the account balance and the daily expenditure. Linear Road is a highly challenging and complicated benchmark due to the complexity of the many requirements. It stresses the system and tests various aspects of its functionality, e.g., window-based queries, aggregations, various kinds of complex join queries; theta joins, self-joins, etc. It also requires the ability to evaluate not only continuous queries on the stream data, but also historical queries on past data. The system should be able to store and later query intermediate results. All these features are also addressed in SRBench. Additionally, SRBench includes Semantic Web specified features, such as inter liked heterogeneous data sets, exploration of RDF graphs and reasoning.

With the growth and availability of many systems supporting RDF/SPARQL, increasing efforts have been made in developing benchmarks for evaluating the performance of RDF stores. The most representative and widely used RDF benchmarks are the Lehigh University Benchmark (LUBM) [21], the Berlin SPARQL Benchmark (BSBM) [8], and the SPARQL Performance Benchmark (SP^{2}Bench) [30]. LUBM, one of the first RDF benchmarks, is built over a university domain in order to mainly evaluate the reasoning capability and inference mechanism of OWL (Web Ontology Language) Knowledge Base Systems. SP^2 Bench uses DBLP [13] as its domain and generates the synthetic data set mimicking the original DBLP data. Currently, BSBM is probably the most popular RDF/SPARQL benchmark that is built around an e-commerce use case where products are offered by various vendors and get the reviews from various customers in different review sites. However, these benchmarks are mostly relational-like, lack heterogeneity or are limited in representing realistic skewed data distributions and correlations. No new features of SPARQL 1.1, such as property path expression have been addressed in these benchmarks. Besides, although one advantage of RDF is the flexibility in sharing and integrating linked open knowledge bases, existing benchmarks solely work with one generated data set without exploiting the knowledge from other linked open data such as DBpedia [17]. Finally, none of the existing benchmarks provides reasoning tasks, a distinguish feature of Semantic Web technology. SRBench has advanced the state-of-the-art of RDF benchmarks by addressing all these features that are hitherto absent.

7 Conclusion

We have introduced SRBench, the first general purpose streaming RDF/SPARQL benchmark, that has been primarily designed to assess the abilities of streaming RDF/SPARQL

[13] `http://www.informatik.uni-trier.de/ley/db/`

processing engines in applying Semantic Web technologies on streaming data. The benchmark has been designed based on an extensive study of the state-of-the-art techniques in both the data stream management systems and the strRS processing engines. This ensures that we capture all important aspects of strRS processing in the benchmark.

Motivated by the study of [17], we have carefully chosen three real-world data sets from the LOD cloud to be used in the benchmark. The benchmark contains a concise, yet comprehensive set of queries which covers the major aspects of streaming SPARQL query processing, ranging from simple graph pattern matching queries to queries with complex reasoning tasks. The main advantages of applying Semantic Web technologies on streaming data include providing better search facilities by adding semantics to the data, reasoning through ontologies, and integration with other data sets. The ability of a strRS engine to process these distinctive features is accessed by the benchmark with queries that apply reasoning not only over the streaming sensor data, but also over the metadata and even other data sets in the LOD cloud. We have complemented our work on SRBench with a functional evaluation of the benchmark on three currently leading streaming RDF/SPARQL engines. The evaluation shows that the functionalities provided by the tested engines are generally limited to basic RDF/SPARQL features over streaming data. There is no single best system yet. We believe that a streaming RDF/SPARQL engine can significantly differentiate itself from other strRS engines by providing more advanced SPARQL 1.1 features and reasoning over both streaming and static data sets.

The natural next step is to run performance and scalability evaluations on the three example strRS engines and probably even other engines. This is a ongoing work. A practical challenge in doing performance evaluation is the verification of query results, given the dynamicity of streaming data and the diversity of the implementing engines.

Acknowledgements. This works was supported by grants from the European Union's 7th Framework Programme (2007-2013) provided for the projects PlanetData (GA no. 257641) and LOD2 (GA no. 257943). We would like to give special thanks to Phuoc-Danh Le for his help on using CQELS.

References

1. D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic. EP-SPARQL: a unified language for event processing and stream reasoning. In *WWW '11*, pages 635–644, 2011.
2. A. Arasu, S. Babu, and J. Widom. CQL: A Language for Continuous Queries over Streams and Relations. In *DBPL2003*, pages 1–19, 2003.
3. A. Arasu et al. Linear Road: A Stream Data Management Benchmark. In *Proc. Of the 30th VLDB Conference*, pages 480–491, Toronto, Canada, 2004.
4. M. Arenas, S. Conca, and J. Perez. Counting Beyond a Yottabyte, or how SPARQL 1.1 Property Paths will Prevent Adoption of the Standard. In *WWW*, 2012.
5. M. Balazinska et al. Data Management in the Worldwide Sensor Web. *IEEE Pervasive Computing*, 6(2):30–40, 2007.
6. D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. Querying RDF Streams with C-SPARQL. *SIGMOD Record*, 39(1):20–26, Mar. 2010.
7. T. Berners-Lee. Linked Data - Design Issues. `http://www.w3.org/DesignIssues/LinkedData.html`, 2009.

8. C. Bizer and A. Schultz. The Berlin SPARQL Benchmark. *Int. J. Semantic Web Inf. Syst*, 5(2):1–24, 2009.
9. A. Bolles, M. Grawunder, and J. Jacobi. Streaming SPARQL - extending SPARQL to process data streams. In *ESWC 08*, pages 448–462, 2008.
10. E. Bouillet, M. Feblowitz, Z. Liu, A. Ranganathan, A. Riabov, and F. Ye. A semantics-based middleware for utilizing heterogeneous sensor networks. In *DCOSS*, pages 174–188, 2007.
11. J. P. Calbimonte, O. Corcho, and A. J. G. Gray. Enabling Ontology-based Access to Streaming Data Sources. In *ISWC*, 2010.
12. CKAN - the Data Hub. `http://thedatahub.org/`.
13. O. Corcho et al. Characterisation mechanisms for unknown data sources. EU Project PlanetData (FP7-257641), Deliverable 1.1, 2011.
14. O. Corcho and R. García-Castro. Five challenges for the semantic sensor web. *Semantic Web*, 1(1):121–125, 2010.
15. DBpedia. `http://wiki.dbpedia.org/`.
16. E. Della Valle et al. It's a Streaming World! Reasoning upon Rapidly Changing Information. *IEEE Intelligent Systems*, 24(6):83–89, November/December 2009.
17. S. Duan, A. Kementsietsidis, K. Srinivas, and O. Udrea. Apples and Oranges: A Comparison of RDF Benchmarks and Real RDF Datasets. In *SIGMOD*, 2011.
18. GeoNames Ontology. `http://www.geonames.org/ontology/`.
19. J. Gray. *The Benchmark Handbook for Database and Transaction Systems*. Morgan Kaufmann, 1993.
20. S. Groppe et al. A SPARQL Engine for Streaming RDF Data. In *SITIS*, 2007.
21. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for owl knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2-3):158–182, 2005.
22. S. Harris and A. Seaborne. SPARQL 1.1 Query Language. W3C Working Draft 05 January 2012, World Wide Web Consortium. `http://www.w3.org/TR/sparql11-query/`.
23. T. Hey, S. Tansley, and K. Tolle, editors. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, Oct. 2009.
24. J. Hoeksema. A Parallel RDF Stream Reasoner and C-SPARQL Processor Using the S4 Framework. Master's thesis, VU University, Amsterdam, the Netherlands, Oct. 2011.
25. D. Le-Phuoc et al. A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data. In *ISWC*, pages 370–388, Bonn, Germany, 2011.
26. D. Le-Phuoc and M. Hauswirth. Linked open data in sensor data mashups. In *Proceedings of the 2nd International Workshop on Semantic Sensor Networks (SSN09)*, pages 1–16, 2009.
27. LinkedSensorData. `http://wiki.knoesis.org/index.php/LinkedSensorData`.
28. J. Pérez et al. Semantics and Complexity of SPARQL. *ACM TODS*, 34(3):1–45, 2009.
29. E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation 15 January 2008, World Wide Web Consortium.
30. M. Schmidt et al. SP^2Bench: A SPARQL Performance Benchmark. In *ICDE*, 2009.
31. J. Sequeda and O. Corcho. Linked stream data: A position paper. In *Proceedings of Semantic Sensor Networks*, pages 148–157, 2009.
32. A. P. Sheth et al. Semantic Sensor Web. *IEEE Internet Computing*, 12(4):78–83, 2008.
33. SRBench wiki. `http://www.w3.org/wiki/SRBench`.
34. The Linking Open Data cloud diagram. `http://richard.cyganiak.de/2007/10/lod/`.
35. O. Walavalkar et al. Streaming Knowledge Bases. In *SSWS*, 2008.
36. K. Whitehouse, F. Zhao, and J. Liu. Semantic Streams: A Framework for Composable Semantic Interpretation of Sensor Data. In *EWSN*, pages 5–20, 2006.
37. Y. Zhang et al. Benchmarking RDF Storage Engines. EU Project PlanetData, Deliverable 1.2, 2011.