Streaming the Web: Reasoning over dynamic data

Alessandro Margara *, Jacopo Urbani, Frank van Harmelen, Henri Bal
Department of Computer Science, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

Received 11 January 2013
Received in revised form
3 February 2014
Accepted 11 February 2014
Available online 2 March 2014

* Corresponding author.
E-mail addresses: a.margara@vu.nl, Alessandro.margara@gmail.com (A. Margara), jacopo@cs.vu.nl (J. Urbani), frank.van.harmelen@cs.vu.nl (F. van Harmelen), bal@cs.vu.nl (H. Bal).

1. Introduction

The Web is highly dynamic: new information is constantly added, and existing information is continuously changed or removed. Large volumes of data are produced and made available on the Web by on-line newspapers, blogs, social networks, etc., not to mention data coming from sensors for environmental monitoring, weather forecast, traffic management, and domain specific information, like stock prices. It has been estimated that every minute on the Internet 600 videos are uploaded on YouTube, 168 millions e-mails are sent, 510,000 comments are posted on Facebook and 98,000 tweets are delivered in Twitter.[1]

In these scenarios information changes at a very high rate, so that we can identify a *stream* of data on which we are called to operate with high efficiency. In the last few years, several researchers and practitioners have proposed solutions for processing streams of information *on-the-fly*, according to some pre-deployed processing rules or queries [1]. This led to the development of various Data Stream Management Systems (DSMSs) [2] and Complex Event Processing (CEP) systems [3,4] that effectively deal with the transient nature of data streams, providing low delay processing even in the presence of large volumes of input data generated at a high rate.

All these systems are based on data models, like for example the well known relational model, which allow only a predefined set of operations on streams with a fixed structure. This allows the implementation of ad-hoc optimizations to improve the processing.

However, the Web provides streams of data that are extremely heterogeneous, both at a structural and at a semantical level. For example, a Twitter stream is radically different from a stream delivered from a news channel, not only because they are stored using different formats, but also because they contain different types of information.

Furthermore, the ability of operating on-the-fly on several of these streams simultaneously would allow the implementation of real-time services that can select, integrate, aggregate, and process data as it becomes available, for example to provide updated answers to complex queries or to detect situations of interests, to automatically update the information provided by a web site or application.

The Semantic Web is an extension of the current World Wide Web, where the semantics of information is encoded in a set of RDF statements. Currently, we are witnessing an explosion of the availability of RDF data on the Web since both public and private organizations have chosen this format to release their public data.[2]

The choice of RDF as data model, in combination with ontological languages (e.g., OWL [5]), enables the implementation of algorithms that can "reason" on existing data to infer new knowledge. Current solutions and technologies for reasoning on RDF data are designed to work on scenarios where changes occur at low volumes and frequencies, and this clashes with the dynamic nature of the streams on the Web.

To bridge this gap, a number of recent works propose to unify reasoning and stream processing, giving birth to the research field of *stream reasoning*. In 2009, stream reasoning was defined as an "unexplored yet high impact research area" [6]. After a few years of research, despite some interesting preliminary investigations in the field, we observe that the stream reasoning research area remains vastly unexplored, both from a theoretical point of view and from the perspective of systems and tools supporting it.

Contributions. In this paper, we first report some example application areas that can benefit from stream reasoning and analyze the requirements they pose. Then, we survey existing approaches in this field, and show why none of them can currently represent a complete answer to all the requirements of various application fields. Starting from this analysis, we isolate some key challenges that need to be addressed to offer full fledged tools for stream reasoning.

Finally, we elaborate a number of possible solutions to overcome the limitations of current approaches. We analyze related research fields to explore whether some topics or solutions, but also algorithms, techniques, and best practices could apply to solve open issues in stream reasoning. In doing so, we intend to illustrate the current state of the stream reasoning research area and to summarize a possible research agenda to further advance in this field.

Outline. The remainder of this paper is organized as follows. Section 2 introduces some example application scenarios for stream reasoning and analyzes the requirements they pose. Section 3 reports a high level introduction to the problem of stream reasoning by describing the research fields that are related to stream processing and reasoning. Next, Section 4 presents a survey of current proposals for stream reasoning and highlights their advantages and limitations. Section 5 extracts the open issues in our current context and Section 6 presents some possible concrete solutions to overcome these issues. Finally, Section 7 provides some conclusive remarks.

2. Motivations for stream reasoning

This section presents some motivations for the need of stream reasoning technologies. It is divided in three parts. In the first part, we present different application scenarios. In the second part, we extract some general requirements that could help identifying the main features expected from stream reasoning. In the third part, we briefly analyze these requirements, with particular focus on their mutual dependencies. Some of the scenarios listed below have already been introduced in previous works on stream reasoning [6,7]. Others are relatively new: for them, the benefits of stream reasoning technologies are discussed for the first time in this paper.

2.1. Motivating scenarios

Semantic Sensor Web. The Semantic Sensor Web (SSW) approach aims at increasing and integrating the communication between

[1] http://www.go-gulf.com/blog/60-seconds.

[2] http://linkeddata.org.

sensor networks [8]. To this end, it introduces semantic annotations for describing: (i) the data produced by the sensors, introducing spatial, temporal, or situation/context semantics; (ii) the sensors and sensor networks that provide such data, to specify details like the measurement precision, battery level, owner or responsible party.

The communities that contribute to this vision are also working on defining suitable ontologies for data and sensors to enable not only the integration of data from multiple sensor networks and external sources, but also to enable reasoning on such data. As an example, a flexible representation of time based on the Time Ontology in OWL has been proposed in [9], while the W3C Semantic Sensor Network Incubator Group [10] developed an ontology to describe sensors and sensor networks.

Sensor data represents an ideal scenario for stream reasoning for several reasons. First, the amount of information collected from sensors on the Web is enormous, and information is often produced at high frequencies. Second, integrating data coming from different sensors (and from different sensor networks) may be necessary in many settings for deriving useful information.

Moreover, sensors provide low level data, like for example temperature readings, and it is challenging to filter out noise and to extract higher level findings, to guide the understanding of complex situations and possibly planning interventions. This task often requires reasoning on time-changing values: for example, it may be relevant to capture the evolution of temperature and wind over time to predict the presence of a tornado.

Some of the scenarios discussed below represent specific application fields for the Semantic Sensor Web. For space reasons, we focus on the domains in which the use of semantic information has been explored in more details. Nevertheless, the requirements identified above generally apply to almost all areas related to sensor networks and to the Internet of Things [11].

Smart cities. Smart Cities represent a specific field where the idea of Semantic Sensor Web can find a concrete application. The final goal of the research in this area is to process and understand the information relevant for the life of a city and use it to make the city run better, faster, and cheaper [12,13].

This poses several challenges, as clearly identified by the experts in the field. First of all, it becomes necessary to deal with significant volumes of data: currently, the sensors in the city of Dublin produce every day about four to six GB of data about the public transport [12], and in the future more sensors will be deployed which will produce more complex data (e.g., HD cameras).

Second, data is extremely dynamic: for example, the position of monitored vehicles can be sensed and updated every few seconds and this demands for efficient algorithms for on-the-fly information management. In this context, the reasoning process must provide enough expressiveness to abstract high level concepts from low level and time annotated data.

Moreover, smart cities require the integration of different data types and sources: as an example, traffic information can be retrieved from sensors, as well as from citizen navigation systems, and from posts on social networks. In this context, it is necessary to consider the veracity and the precision of information.

Smart grids. Smart grids [14] represent another scenario that requires data monitoring and integration, situation detection, and (partially or completely) automated decision making. The goal of smart grids is to make current energy grids more efficient and sustainable by collecting and interpreting information coming from different stake holders, e.g., energy producers, grid operators, or appliance manufacturers. Similar to smart cities, smart grids require integration, management of large volumes of dynamic data, and on-the-fly analysis of time-annotated data to extract high level knowledge and to timely provide support for decision.

Remote health monitoring. Remote health monitoring [15,16] aims at generating automated and personalized systems for remote patient monitoring. In particular, these systems focus on collecting information from multiple sources (e.g., sensors for monitoring the heart rate or blood pressure), applying a reasoning step to understand the context or situation of the patient and to guide the decision making process. Also in this context, one of the main challenges is the integration of data from multiple sources: e.g., it may be useful to know the current activity of the patient to understand if the measured heart rate is too high.

Nanopublications. Nanopublications [17,18] represent a new and vastly unexplored field of research. They have been proposed as a way to represent the key findings of scientific research and publications, using models and languages that allow a more refined representation of the meaning of the data. In this way the data can be understood and processed more efficiently by computers, enabling automatic detection of inconsistencies, classification of existing literature, and analysis of provenance.

Differently from the previous scenarios, nanopublications do not (currently) introduce strict real-time constraints. However, in this domain there is a significant amount of background knowledge that can be used to guide the validation of new information published on the Web, for example on Wikipedia or on blogs of experts. Moreover, nanopublications can facilitate the integration of scientific and experimental data so that they can be exploited to guide and support the process of discovering new findings.

Drug discovery. Drug discovery represents a concrete field of research in which semantic data is being used for guiding the discovery process. For example, this has been investigated in the Open PHACTS project [19,20], which aims at reducing the communication barriers between different universities and companies by providing tools and services for integrating their data. While drug discovery does not introduce strict real-time constraints, efficiency remains a key requirement. Drug Discovery tests are usually conducted by multiple parties with high costs. Because of this, it is important to know as soon as possible whenever a part of drug discovery test has failed in order to stop or adapt other tests and preserve further costs.

Abstracting and reasoning over ship trajectories. Another field in which the use of complex reasoning over time-annotated data has been investigated is Maritime Safety and Security [21], where sensor data from ships is combined with external data to derive new knowledge. This scenario uses temporal relations and patterns of changes to extract higher level knowledge from low level data collected from sensors.

In this context, it is particularly important to define methods that can correctly handle the uncertainty that comes from the input. Uncertainty is inevitably linked to sensor data and may appear in different forms, from incorrect measurements to data loss, and it is particularly important in this scenario since the communication often takes place over unreliable channels. Complex forms of modeling, which involve advanced analysis of time annotated data received from ships, may help to infer correct knowledge even in presence of temporary data loss. Similarly, the integration with other sources of information may enable to re-construct missing data and to validate measurements coming from the sensors.

Currently, reasoning over ship trajectories has only been performed on historical data, analyzing the time-annotated data received from sensors after it has been collected and stored. Moving to on-the-fly processing of data as it is generated would allow for immediate detection (and possibly reaction) of anomalies such as, for example, pirate attacks or engine failures.

Analysis of social media and mobile applications. The advent of social networks, blogs, mobile services and applications provides large volumes of data that can be seen as "sensors" of people moods, interests, relations, etc. Semantic analysis of social media [22,23] extends traditional analysis based on graphs enriching the connections between people and concepts with semantic annotations.

One of the goals of the analysis of social media is to capture hidden relations between people and concepts. In this scenario, it is interesting to detect not only the current situation or context, but also the historical evolution of relations over time. This problem calls for mechanisms and techniques that can reason over time-changing relations, and can compare the current information with historical data. Finally, extracting knowledge from social media and mobile applications presents additional complexity due to noise, possible inconsistencies and ambiguous representation of the data.

Some experiments on the use of social media analysis have been reported in [24], where the authors focus on the processing of posts to Twitter to extract new information (e.g., how the mood changes during the day, which are the trending topics) during some large-scale events (London Olympic Games 2012 and Milano Design Week 2013).

2.2. Requirements of the use cases

From the analysis of the previous scenarios, we can extract some common aspects, which can be translated into requirements demanded to stream reasoning tools. In this section, we describe them and highlight the main challenges that they introduce. Table 1 summarizes the requirements and shows their connection with the application scenarios.

Integration. In all the scenarios we analyzed, the integration of information that comes from *multiple sources* represents a key requirement. This motivates the usage of data models like RDF that allow semantic integration. Data integration poses both conceptual and technical challenges. The firsts are related with issues such as data provenance and trust, while the seconds with other aspects such as the management and combination of multiple data format in a single system. Moreover, all the scenarios might require the integration of the data coming from the stream with some *background knowledge* that describes the application domain. For example, the remote health monitoring use case requires that the streaming data collected from the patients is combined with other background knowledge about symptoms and diseases. This integration is challenging, since retrieving and analyzing large volumes of background data during stream processing can be particularly expensive with current technologies.

Time management. Time plays a central role in almost all the scenarios that we considered. This demands for a suitable *data model* where data items can be annotated with their time of occurrence and validity, and where it is possible to express data changes. Moreover, the management of time often requires a novel *processing model*, where the time relations between data items become first class objects. For instance, in the smart cities scenario, it is necessary to look at the time series of vehicles positions to predict and prevent traffic jams. Similarly, in the context of social media analysis, new knowledge can be derived from the dynamic evolution of people relations. Introducing processing abstractions for time-annotated and dynamic data does not only represent a theoretical challenge, but also introduces engineering issues that are related to an efficient implementation of such abstractions.

Distribution. Many of the previous scenarios acquire the input data from sources that are geographically distributed. This introduces significant challenges in terms of synchronization, out-of-order arrival of information, and communication loss. These issues become even more relevant in presence of a processing model where time and timing relations represent key concepts. Moreover, in presence of large volumes of data or resource-constrained devices, communication can easily become a performance bottleneck. To alleviate this problem, *distributed processing* may become necessary. For example, filtering irrelevant information at its source may reduce the communication overhead in sensor networks.

Big data management. All scenarios require processing of large amounts of data. On the one hand, this is due to the integration of multiple sources, and from the analysis of rich background data. On the other hand, in most scenarios, new information is produced at high rate. Consider for example the Smart Cities scenario, where reasoning may take into account the position of cars, their speed, information coming from fixed sensors on the road, data from traffic applications, etc., which are continuously changing over time. This introduces significant challenges in terms of data processing, communication, storage and retrieval, but also selection, filtering of irrelevant information, and veracity.

Efficiency. The requirement for processing efficiency is strictly related with the big data management. On the one hand, most of the scenarios demand for a high processing *throughput*, to cope with the large amount of data that is being produced. Furthermore, some scenarios also demand for *low latency* processing, to timely generate new results. For example, in the case of Semantic Cities, car accidents or traffic anomalies need to be promptly detected and communicated, enabling counter actions.

Expressivity. All scenarios target at deriving high level knowledge from large volumes of low level information. Defining a suitable processing model is probably one of the main challenges that the research on stream reasoning needs to face.

In Section 6 we describe this issue in more detail. Here, we recognize three main requirements for a suitable processing model. First of all, all scenarios involve some form of reasoning, which must be supported by the processing model. Secondly, since most scenarios need to capture the dynamicity of the data over time, it is required that the processing model offers abstractions and operators for dealing with time-changing data.

Finally, extracting high level knowledge often requires operators for data computation and transformation, which must be supported by the processing model. For example, the aggregation of readings from multiple sources is often used in sensor networks to compute the average temperature in a certain monitored area, while computation of users statistics could be relevant in social media and mobile applications analysis.

In Section 6 we describe this issue in more details. Here, we recognize three main requirements for a suitable processing model. First of all, all scenarios involve some form of reasoning. Second, since most scenarios need to capture the dynamicity of the data over time, it is required that the processing model offers abstractions and operators for dealing with time-changing data. Third, extracting high level knowledge often requires operators for data computation and transformation. For example, aggregation of readings from multiple sources is often used in sensor networks, e.g., to compute the average temperature in a certain monitored area, while computation of users statistics could be relevant in social media and mobile applications analysis.

Uncertainty management. A key requirement identified in many application scenarios is the management of the uncertainty associated to data. Uncertainty can appear at different levels: data may be imprecise or missing (e.g., imprecise measurements or data loss in sensor networks), contradictory (e.g., in the scenario of nanopublications, where different papers can introduce contradictory statements), noisy or unstructured (e.g., in social media analysis data is extracted from the content of users publications). This demands for a suitable model that takes into account the different sources of uncertainty, and considers them during the computation.

Historical data management. Beside processing streaming data, most applications need to manage *historical data*, i.e., time-annotated information about previous states of the environment under analysis (for example, the history of temperature readings from a certain area). In particular, they need to store historical data and make it available for query and retrieval. This can be used, for

Table 1
Analysis of requirements for application scenarios.

Scenario	Integration		Time manag.		Distribution		Big data manag.	Efficiency		Expressivity			Uncertainty manag.	Historical Data	Quality of service
	Different sources	Streaming/background data	Data model	Processing model	Distrib. sources	Distrib. processing		High throughput	Low latency	Reasoning	Temporal operators	Data transformation			
Semantic sensor	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Smart cities	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Semantic grids	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Health monitoring	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓	
Nanopublications	✓	✓			✓		✓			✓		✓	✓		
Drug discovery	✓	✓			✓	✓	✓		✓	✓		✓	✓		
Ship trajectories	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	
Social media	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	

example, to identify trends, to extract statistics, or to compare previous and current information to detect anomalies. The latter is particularly challenging, since it requires to access and analyze potentially large amounts of historical data on-line, during the processing of the streaming data.

Quality of service. Despite the commonalities identified above, every application scenario presents its own requirements. Stream reasoning systems need to provide enough flexibility to satisfy heterogeneous needs through explicit Quality of Service policies. As an example, dropping some information to reduce the computation and hence returns results with lower delay may be acceptable in some scenarios, and perhaps even desirable, while in other scenarios incomplete information may be inadmissible. Different application fields demand for different ontologies and reasoning complexity and processing tools should provide flexible mechanisms to easily adapt their behavior to different requirements.

2.3. Analysis of requirements

The scenarios proposed in Section 2.1 are rather heterogeneous. Because of this, the requirements identified in Section 2.2 are not equally relevant for all of them. For example, efficiency is of primary importance in sensor applications, which demand for on-the-fly evaluation of the input. Conversely, nanopublications and social media analysis do not impose strict real-time requirements.

We observed that all the application fields demand for some form of reasoning. However, the complexity of the reasoning task may vary significantly from application to application. It remains an open question to identify the reasoning capabilities and expressiveness required in each scenario.

Similar arguments hold for data management: different applications deal with different volumes and different update rates and stream reasoning technologies are called to be applied in this large space of problems and scenarios. Because of the trade-offs discussed above, it is unknown whether it is possible (or beneficial) to develop a single solution to satisfy all of them, or if different design models, algorithms, and implementations are needed to target specific parts of this space. We will come back to this issue in Section 6, after presenting current solutions in the area and discussing future research directions.

3. Survey: background

This section introduces the research areas strictly related to stream reasoning, presenting state of the art systems for stream processing (Section 3.1), reasoning (Section 3.2), and other related work (Section 3.3).

3.1. Stream processing

To present stream processing systems, we refer to established models from the literature [25,26,1]. Fig. 1 shows the general architecture of stream processing systems. They are designed to process streams of data, generated by a number of *sources* to timely produce new results for the interested consumers. The computation is defined in terms of a set of *rules*, or *queries*, which are defined by the user and deployed into the system.

Stream processing systems reverse the interaction model of traditional DBMSs. DBMSs store data and allow consumers to submit one-time queries to retrieve it. Conversely, stream processing systems allow consumers to submit rules that are continuously evaluated to produce new results as new input data is received. The data items present in the stream are usually annotated with timestamps that indicate timing or ordering relations. In some systems, these annotations are used to identify time patterns, like sequences or repetitions of specific elements.

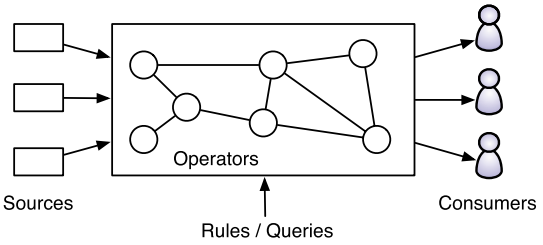

Fig. 1. Abstract model of stream processing systems.

As shown in Fig. 1, stream processing systems organize the computation into a graph of primitive operators. Depending on the actual implementation, these operators can be physically deployed on a single node, or on multiple connected nodes. The main focus of stream processing system is to support large volumes of input data produced at high rate, and to offer fast response time to the consumers.

Existing stream processing systems differ from each other on a wide range of aspects, including the data model used to specify the input, the language used to define processing rules, and the processing techniques adopted. In this section, we adopt the same classification criterion used in [1], and divide these systems into two classes: *Data Stream Management Systems* (*DSMSs*), which have been developed by the database community and *Complex Event Processing* (*CEP*) systems, which have been developed by the community working on event based systems.

3.1.1. Data Stream Management Systems

Data Stream Management Systems [2] inherit their data and query model from traditional DBMSs. Indeed, despite several systems have been proposed [27–30], almost all of them follow the design line of the pioneer Stream system [31]. In these system, data is represented using the relational model, and queries are expressed with declarative languages that are derived from SQL. Such languages include *windows* to isolate portions of the (unbound) input streams and to convert them into relational tables. For example, windows are commonly used to select only the most recent elements in a stream, based on their number (*count-based windows*), or on their associated timestamp (*time-based windows*). After windowing, the input data is processed using relational operators, and the output is converted again into a stream using specific operators called *relation-to-stream operators*. Consumers can decide to stream the complete results of the last evaluation (*snapshot semantics*) or only the differences with respect to the previous evaluation (*incremental semantics*).

As an example, consider the query below, expressed in the CQL [32] language adopted by the Stream [31] system.

```
Select IStream(*)
From S1 [Rows 5], S2 [Range 10 sec]
Where S1.A = S2.A
```

This query considers two streams S1 and S2. First, it uses windows operators to isolate the portions of the stream to be evaluated during processing. It considers a count-based window for S1, which always includes the last 5 items, and a time-based window for S2, which includes all the information items received in the last 10 s. Then, the query uses the relational operator *join* to join together all the items in S1 and S2 that share the same value of A. Finally, the `IStream` operator outputs new results generated during the last evaluation (incremental semantics) into a new stream.

The main advantage of DSMSs consists in the usage of the well known and widely adopted relational model for representing and processing the data. However, this comes at a cost: processing is only allowed inside the portions of input streams identified by windows. This makes it difficult to express and capture complex temporal patterns over the input streams.

3.1.2. Complex Event Processing systems

Complex Event Processing (CEP) systems take a different approach than DSMSs. Input data items represent timestamped notifications of event occurrences, often encoded as records of key–value pairs. For example, an event notification may represent a temperature reading from a sensor at a specific instant of time, and it can include values like the actual temperature, the position of the sensor, or the status of its battery. Events observed from the external environment are called *primitive* events. Processing rules define the occurrence of higher level *composite* events as (temporal) patterns of primitive ones. In other words, processing rules define rewriting policies that translate patterns of input events into one or more output events.

As an example, consider the rule below, expressed in the TESLA [33] language, used by the T-Rex [34] CEP system.

```
define    Fire(area: string, measuredTemp: double)
from      Smoke(area=$a) and
          last Temp(area=$a and value>45)
          within 5 min. from Smoke
where     area=Smoke.area and measuredTemp=Temp.value
```

This rule defines the composite event `Fire`. More in particular, the rule defines a pattern including a *sequence* of two primitive events: `Temp` and then `Smoke`. It also includes constraints on the *content* of primitive events (they have to come from the same area, as expressed by the parameter `$a` and `Temp` must include an attribute `value` which is greater than 45), and on the *occurrence time* of events (`Temp` and `Smoke` must occur within 5 min from each other).

If we compare DSMSs with CEP systems, we notice that the former, as evolution of traditional DBMSs, are designed to reorganize and retrieve data. Their main processing building blocks are the relational operators, which enable rules to *transform* (select, filter, join, etc.) static and streaming input it into new output. Conversely, CEP systems were designed to deal with the specific problem of *defining* new (higher level) events or situations starting from the primitive events (raw data) observed. For this reason, CEP languages rely on explicit operators for the definition of (temporal) patterns, like conjunction, disjunction, sequences, and repetitions of event notifications. The aim of a CEP system is to *detect* occurrences of composite events starting from the definitions provided in rules.

Several CEP systems have been developed in the last few years [35,36,34,37–41]. One of the main goals of these systems is to execute the rules efficiently; accordingly, most of them trade expressiveness for response time.

3.2. Reasoning

The Semantic Web is a recent research area that studies how we can inject "meaning" (or better semantics) in the data that is available on the Web, so that machines can process it more efficiently. For example, machines could process semantically-enriched data to infer new implicit information, or detect inconsistencies that are not immediately visible in the input. To this end, the W3C organization has released a number of specifications to define data models or ontology languages to represent the data. The most notables are RDF [42], and OWL [5], which are nowadays the common denominator for most of the research in the area.

A combined usage of RDF and OWL (or others like RDFS [43] or the Horst fragment [44]) enables the representation of semantically enriched data that can be used by applications to infer implicit knowledge. This inference process is commonly referred to as *reasoning*. Reasoning can be performed for several purposes. For example, it can be used to execute automatic consistency checking when integrating multiple sources of knowledge, or to classify new information, or enrich query answers with new knowledge.

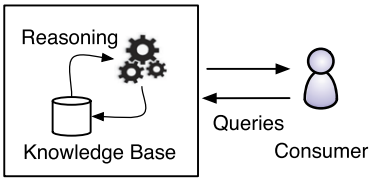

Fig. 2. Abstract model of reasoning systems.

If compared to the processing performed by stream processing systems, reasoning is more computationally expensive. Depending from the complexity of the ontology language adopted, reasoning can even become undecidable. To alleviate these problems, the W3C has defined several profiles [45] of OWL that reduce the complexity of reasoning making it more tractable and scalable to large amount of input information.

Several works have tried to scale the reasoning process on larger inputs using a variety of computer architectures, like clusters [46], supercomputers [47], GPUs [48], or using state of the art database engines [49,50]. Despite these efforts, reasoning remains a complex task. Because of this, it has been applied to static data that is assumed not to change, or to change very infrequently over time. Fig. 2 shows the abstract architecture of a reasoning system: reasoning is applied to a static knowledge base, to derive implicit information. Consumers can interact with such knowledge base by issuing one-time queries, usually expressed in the SPARQL [51] language.

3.3. Related work

While stream reasoning mainly inherits concepts and techniques from the two aforementioned fields of stream processing and reasoning, there are other related research areas that can contribute to its development. We briefly describe them below.

Active database systems. Traditional DBMSs are completely passive. They return data only when explicitly queried by the consumers. Active database systems [52] overcome this limitation by implementing reactive mechanisms. In particular, they enable the consumers to install Event Condition Action (ECA) rules, which specify actions to be performed whenever an event is observed and certain conditions are met. Usually, such reactive mechanisms are adopted to perform consistency check on the internal state of the database. For example, they are used to express actions to be performed in case some consistency constraints are violated. Interestingly, some works in the area of active DBMSs proposed more expressive formalisms than ECA rules, which can support the detection of complex (temporal) patterns [53].

Temporal database systems. Temporal database systems [54] have been proposed to store and query data that involves time. In particular, data items have usually an associated validity time, which specify the period of time in which they are valid. This allows the update of the information stored in the database over time and to store historical data.

Rule-based systems. Rule-based systems [55] target at manipulating information based on derivation rules. They are strictly connected with reasoning and event processing systems. Indeed, rule-based processing is the underlying mechanism adopted in many reasoning tools. Moreover, a few languages based on derivation rules have been proposed in the domain of CEP [56,57].

Deductive database systems. Deductive database systems [58] combine logic, rule-based programming with relational database making it possible to derive implicit knowledge from explicit information stored in the database. In deductive databases, rules are usually expressed in Datalog [59], a subset of Prolog that targets at enabling efficient evaluation over large databases.

Table 2
Classification of existing solutions.

System	Continuous queries	Background data	Time model	Reasoning	Temporal operators	Data transfor.	Uncertainty manag.	Historical data	Quality of service	Parallel/distributed processing
C-SPARQL [60,61]	✓(periodic)	✓	Single TS/occurrence	Not implemented		✓				
IMaRS [62,63]	✓(periodic)	✓	Single TS/occurrence	Transitive property		✓				
TrOWL [64,65]	✓	✓	Single TS/occurrence	OWL2 DL—synt. approx.						
Inductive stream reasoning [66]	✓	✓	Single TS/occurrence	RDFS/OWL2 RL (discussed, not implemented)		✓				
C-SPARQL on S4 [67]	✓		Single TS/occurrence	RDFS subset		✓				✓
CQELS [68]	✓	✓	Single TS/occurrence			✓				✓
Streaming-SPARQL [69]	✓	✓	Single TS/occurrence							
Streaming Knowledge Bases [70]	✓	✓	Single TS/occurrence	RDFS/OWL2 subset						
Sparkwave [71]	✓	✓	Single TS/occurrence	RDFS subset						
EP-SPARQL [72]	✓	✓	Interval/occurrence	RDFS subset	✓	✓		✓		
Answer set programming [73]	✓	✓	Single TS/occurrence	ASP reasoning	✓	✓				
TA-SPARQL [74]		✓	Single TS/occurrence			✓		✓		
Time RDF [75]		✓	Various/occurrence			✓		✓		

4. Survey: systems review

This section reviews existing contributions in the area of stream reasoning. First, we present in Section 4.1 the classification criteria that we adopted. Then, we analyze in Section 4.2 each system in detail.

4.1. Classification criteria

While analyzing existing work, we mainly focused on the key properties that we derived from the analysis of application requirements in Section 2.

- *Continuous queries.* This criterion defines whether the system supports continuous queries, i.e., queries that are registered and produce new or updated results as new input data becomes available. We distinguish between systems that enable query evaluation to be synchronized with the arrival of new information and systems that only enable periodic evaluation of queries with a predefined frequency. This property is fundamental to support streaming information and on-line processing, thus satisfying the requirement for *integration* of streaming and background data.
- *Background data.* Defines whether the system can consider background data during processing, i.e., non time-annotated information about the domain of analysis. Like the previous property, also this one is necessary to satisfy the requirement for *integration* of streaming and background data.
- *Time model.* Defines how the system annotates the data with temporal information (e.g., using a single timestamp, defining a point in time, or multiple timestamps, defining intervals of time) and which is the semantics of this information (e.g., time of occurrence, time of validity). This property maps to the requirement for *time management*.
- *Reasoning.* Defines whether the system can perform reasoning over *both* streaming information and background data. Whenever possible, we specify the kind of reasoning that is supported. This property maps to the requirement for *expressivity*.
- *Temporal operators.* Defines whether the system offers explicit operators for capturing the dynamicity and evolution of information over time, e.g., for detecting time series or sequences. This property maps to the requirements for *expressivity* and *time management*.
- *Data transformation.* Defines whether the system offers processing abstractions for manipulating the input information, e.g., operators for aggregation. This property maps to the requirement for *expressivity*.
- *Uncertainty management.* Defines whether the system supports modeling, evaluation, and propagation of uncertainty.
- *Historical data.* Defines whether the system supports storage and retrieval of time-annotated historical data.
- *Quality of service.* Defines whether the system enables consumers to define QoS policies for processing.
- *Parallel/distributed processing.* Defines whether the system offers support for parallel and distributed processing to improve scalability or response time. This property maps to the requirements for *big data management*, *distribution*, and *efficiency*.

Notice that Table 2 does not include any indication of the level of performance and scalability of the systems under analysis. Indeed, as recognized in the literature [76–79], a precise evaluation and comparison of existing systems is extremely complex. Nevertheless, whenever possible, we will present and discuss existing case studies, applications, and assessments of the performance of the systems.

4.2. Analysis of existing systems

Table 2 summarizes our analysis of existing systems and shows their relations with the aforementioned classification criteria. We organize our presentation into three main classes.

First, we consider *semantic stream processing* systems: we include in this class all the systems that inherit the processing model of DSMSs, but consider semantically annotated input, namely RDF triples, and define continuous queries by extending SPARQL.

Then, we consider *semantic event processing* systems: these are systems that pose their roots in a processing paradigm that is more

similar to that of CEP systems, and offer operators for (temporal) pattern detection as the main building blocks for computation.

Third, we consider systems for *time-aware reasoning*. They provide abstractions for reasoning on time annotated data, but still lack stream processing capabilities.

Finally, we also include an additional section for existing surveys and vision papers on stream reasoning and connected topics.

4.2.1. Semantic stream processing

C-SPARQL. C-SPARQL (Continuous SPARQL) [60,61,80] is among the first contributions in the area of stream reasoning and is often cited as a reference in the field. It is a new language for continuous queries over streams of RDF data with the declared goal of bridging the gap between the world of stream processing systems, and in particular DSMSs, and SPARQL.

The distinguishing features of C-SPARQL are (i) the support for timestamped RDF triples, (ii) the support for continuous queries over streams, and (iii) the definition of ad-hoc, explicit operators for performing data aggregation, which is seen as a feature of primary importance for streaming applications.[3] The results of C-SPARQL queries are continuously updated as new (streaming) data enters the system.

When evaluating a query, C-SPARQL can process (and combine together) both background data stores and streams. It borrows the concept of windows from DSMSs to capture the portions of each stream that are relevant for processing. In C-SPARQL, a window always selects the most recent items from a stream, and can be either count-based (selecting a fixed number of elements) or time-based (selecting a variable number of elements which occur during a given time interval). Consumers can specify additional policies for the advancement of windows: they can be sliding (i.e., they move every time a new element is received from the stream) or tumbling (i.e., they always include new information elements, ensuring that every element is considered at most once during processing). Query evaluation is performed as in traditional SPARQL, but considering only the data that is present in the current windows. It is worth noting that in C-SPARQL each query has a pre-defined and fixed evaluation frequency, which can specified by the consumers. As a consequence, query evaluations are decoupled from the arrival of new data from the stream. This decoupling has been identified as a significant limitation of the processing model of C-SPARQL [77]. Since its evaluation model is directly inherited from DSMSs, C-SPARQL does not include any explicit operator to capture temporal patterns over input elements. Although C-SPARQL allows to define queries that consider time by directly accessing the timestamp of each information item, this is possible only inside the boundaries of windows.

An execution engine for C-SPARQL has been implemented as a framework that combines existing RDF repositories (e.g., Sesame [81]) and DSMSs (e.g., Stream [31]). Although the proposed processing model enables reasoning on streaming data, this is currently not supported in the execution engine. As observed in the literature [68], the lack of deep integration reduces the possibilities for optimization and for parallelization and distribution.

The authors present C-SPARQL through a Urban Computing use case, which exploits information coming from sensors and from mobile phone applications to extract and present new knowledge. In [82], they reference a comprehensive list of requirements for information processing in Urban Computing.

The evaluation of the existing execution engine is currently provided for a limited number of queries and addresses relatively small scale scenarios, with up to 100,000 triples in static stores and windows, showing processing times that are in the order of tens of milliseconds. These results are controversial. A recent analysis on the performance of existing stream reasoning systems [77] emphasizes some limitations of the C-SPARQL execution engine, measuring low execution throughput (often below one execution per second) and scalability, both in terms of input size and in terms of number of queries.

IMaRS. The authors of C-SPARQL provided another contribution to the area of stream processing in [62,63], by presenting an algorithm for maintaining the materialization of ontological entailment in the presence of streaming information, IMaRS (Incremental Materialization for RDF Streams). While incremental reasoning was already explored in the literature [83,84], IMaRS represents a novelty. It relies on the streaming nature of the input and on the specific data and processing models of C-SPARQL to compute the expiration time of streaming RDF triples based on the windows of deployed queries.

By annotating each RDF triple with its expiration time, IMaRS reduces the amount of computation that needs to be performed to update the results of reasoning. The algorithm selectively identifies and drops expired statements, and computes new materializations in an incremental way, exploiting the previous results that are still valid.

Although very interesting, IMaRS relies on the strong assumption that the expiration time of each triple can be pre-computed, which limits its applicability. For example, this assumption is not longer valid in presence of count-based windows, where the expiration time of triples depends on the frequency of arrival of new triples.

The proposed approach was presented through a Urban Computing use case. An empirical evaluation was conducted using the transitive property on a small dataset (the precise size was not specified, but all the information could fit into 2 GB of main memory). Under this setting, IMaRS brings significant advantages when the percentage of the background knowledge that changes is relatively small, while its costs overcome the benefits when more than 13% of the background knowledge changes. This is due to the overhead of computing the expiration times of incoming triples and annotating them.

TrOWL. TrOWL [64,65] is another tool for incremental reasoning. Compared to IMaRS, TrOWL presents two main distinctive features: (i) it offers support to more complex ontology languages, covering the expressive power of OWL2-DL; (ii) it does not rely on fixed time windows to predict the expiration time of streaming information. The main idea behind TrOWL is the use of syntactic approximation to reduce reasoning complexity. Syntactic approximation ensures that all derived knowledge is correct, i.e., it ensures soundness, but not completeness of reasoning.

To simplify retraction of information, TrOWL keeps traceability relations between deriving facts and derived facts. The authors provide experimental evidence of the benefits of their approach while modifying increasing portions of the knowledge base. They use the Lehigh University Benchmark (LUBM) ontology [85], which provides automatic generation of ontologies about universities. In particular, it enables to control the size of the ontology by manipulating some parameters, e.g., the number of universities and the number of departments they include. The authors perform their experiments in a small scenario, considering one university with 15 departments (about 100,000 triples).

Unfortunately, it is hard to compare the results of TrOWL with IMaRS. The authors of TrOWL consider updates from 20% to 80% of the ontology and measure a processing time that varies from 20% to 70% with respect to the naive approach of recomputing all the derivations. On the other hand, IMaRS focuses on smaller updates (up to 20% of the ontology), presenting large advantages when less

[3] Ad-hoc aggregates were abandoned after the introduction of SPARQL 1.1 aggregates.

than 10% of the ontology is changed, but becoming slower than the naive approach soon after.

Inductive stream reasoning. An additional and noteworthy extension to the C-SPARQL model is presented in [66], where the authors focus on *inductive* stream reasoning, which consists in mining of large portions of data, and applying statistical and machine learning techniques to extract new knowledge.

The authors propose to combine inductive with deductive reasoning to increase accuracy. The technique has been applied and validated on a real scenario deriving from social media analysis [86], showing the accuracy of the reasoning algorithm in this context. The authors also present some performance results, showing that the proposed approach exhibits low processing delays (constantly below 10 ms) which outperforms traditional SPARQL query engines. The experiments are conducted on a small scale scenario, where evaluation windows include at most 2500 triples.

C-SPARQL on S4. In [67], the authors implement partial RDFS reasoning and part of the C-SPARQL query language on the S4 streaming platform, which enables to split the processing load over multiple machines to increase the overall system throughput.

More in particular, the authors implement operators for triple selection, filtering, join, projection, and aggregation. To improve the parallelization, the authors only consider time-based windows, while they do not implement count-based windows. Indeed, for a time-based window w, it is possible to evaluate whether a triple is included in w in a distributed way, without collecting all the input triples in a single processing node.

The solution has been tested on up to 32 nodes, using the Berlin SPARQL Benchmark (BSBM) [87]. Some preliminary results are presented, exploiting four queries with different level of complexity, ranging from a simple passthrough query (selecting every triple in input) to more complex queries requiring multiple joins. In this setting, the authors measure an input throughput of more than 150,000 triples per second when no reasoning is considered, and of more than 60,000 triples per second with RDFS reasoning. Most significantly, the performance scales linearly when moving from one to eight processing nodes, but the advantages decrease with more nodes.

CQELS. In [68], the authors propose a new language, CQELS, and an accompanying query engine for combining static and streaming linked data [88,89]. Similar to C-SPARQL, CQELS adopts the processing model of DSMSs, providing windowing and relational operators together with ad-hoc operators for generating new streams from the computed results. Differently from C-SPARQL, CQELS offers a processing model in which query evaluation is not periodic, but triggered by the arrival of new triples.

The distinctive difference of this solution with respect to C-SPARQL is in the processing engine, which strictly integrates the evaluation of background and streaming data, without delegating them to external components. This makes it possible to apply query rewriting techniques and optimizations well studied in the field of relational databases.

The authors present a detailed experimental evaluation, where they compare the engine of CQELS against C-SPARQL and EP-SPARQL (see below). In particular, they use simulated DBLP datasets created with the SP^{2}Bench [90] and test the scalability on the input size (number of triples) and on the size of the window under analysis. The measured results show the benefits of integration and query optimization, with CQELS providing execution times that are up to more than 1000 times lower than the other approaches. All the examples and results discussed in the paper, however, only describe query answering and do not take into account reasoning over streaming data. Additional results have been published in [77], with a more detailed comparison of the processing models and performance of existing stream processing systems, showing even larger advantages of CQELS over C-SPARQL, especially in terms of scalability. However, these results are controversial, since recent publications [78,79] highlights some differences in the semantics of CQELS and C-SPARQL. According to the authors, such differences make the performance results incomparable.

The number of studies devoted to comparing existing solutions, together with the heterogeneous conclusions they derive, is a clear indication of the difficulties in assessing the performance of stream reasoning systems. We will discuss the problem in more details as part of our research agenda in Section 6.

Recently, the authors of CQELS introduced a new implementation explicitly designed for cloud environments [91], called CQELS cloud. CQELS cloud translates the set of continuous queries deployed in the system into a network of operators. A central coordinator maps operators to nodes, trying to minimize network traffic: for instance, operators that consume the same input data are deployed on the same node, if it provides enough processing resources. The implementation includes algorithms for incremental processing of complex operations, like aggregates and joins. Furthermore, CQELS cloud also supports elasticity, allowing nodes to join or leave, and re-assigning operators to nodes accordingly. The authors evaluate CQELS cloud using different workloads and show how the throughput of the system scales linearly with the number of processing nodes. However, they only consider matching, join, and aggregate operators and do not include any reasoning task.

Streaming-SPARQL. Streaming-SPARQL is another extension of SPARQL designed for processing streams of RDF data which is presented in [69]. The main contributions of this work are theoretical. In particular, the authors mainly focus on the specification of the semantics of Streaming-SPARQL using temporal relational algebra and provide an algorithm to automatically transform SPARQL queries into this new extended algebra.

Unfortunately, no experimental evaluation of the approach is provided. Although promising, the approach currently supports only the transitive property, and has been tested on a small scenario that fits in a single machine with only 2 GB of main memory.[4]

Streaming Knowledge Bases. Streaming Knowledge Bases [70] is built on top of the TelegraphCQ [92] DSMS and provides reasoning using a subset of RDFS and OWL reasoning over streaming RDF triples. The key aspect of this proposal is its ability to precompute and store some inference to reduce the computational effort, and consequently the delay, during the evaluation of the queries. The current implementation has been tested on relatively small datasets, with up to 60,000 resource entries. The authors plan to handle more complex reasoning in the future.

Sparkwave. Sparkwave [71] is a system designed for high performance on-the-fly reasoning over RDF data streams. It trades complexity for performance: in particular, Sparkwave poses severe limitations to the size of the background knowledge, which must fit into the main memory of a single machine; moreover, it operates over a pre-loaded RDF schema and provides limited reasoning functionalities. Sparkwave implements a variant of the RETE algorithm [93], in which a pre-processing phase is used to materialize derived information before performing pattern matching. The portions of data considered during the processing are isolated through traditional windowing mechanisms, similar to those used by DSMSs and C-SPARQL.

The authors offer a preliminary evaluation of the approach using the Berlin SPARQL Benchmark (BSBM) [87]. They consider up to 100,000 triples in the background knowledge and consider four queries of increasing complexity (requiring from simple selection

[4] The source code, used in a case study in a sensor network scenario, is available online at http://code.google.com/p/semanticstreams/wiki/SPARQLStream.

to multiple joins). In this setting, Sparkwave provides a higher throughput than C-SPARQL and CQELS (up to more than 100,000 triples per second), consuming less memory (at most 1 GB, less than a half with respect to C-SPARQL). However, as already mentioned, the current version of Sparkwave introduces several limitations with respect to expressiveness: it only supports graph pattern detection and does not include neither logical operators (disjunctions and negations), nor arithmetic operators, nor temporal operators.

4.2.2. Semantic event processing

EP-SPARQL. EP-SPARQL [72] is a unified language for event processing and reasoning. To the best of our knowledge, it is currently the only solution that inherits the language constructs and processing model of CEP systems. Similarly to C-SPARQL, EP-SPARQL provides windowing operators (both count and time-based) for isolating portions of the input streams which will be processed by the system. However, differently from C-SPARQL the main building blocks of the EP-SPARQL language are represented by a set of logical and temporal (sequence) operators that can be combined to express complex patterns of information items. Another notable difference of EP-SPARQL with respect to C-SPARQL is in the data model and consists in the way time is associated to RDF triples. While C-SPARQL associates one timestamp to each triple, representing a single point in time—point semantics, EP-SPARQL adopts two timestamps, which represent the lower and upper bound of the occurring interval—interval semantics. This reflects on output triples, whose occurrence intervals are computed from the input elements that contributed to their generation.

The idea is promising and makes it easy to write complex patterns involving content and time constraints on the input RDF triples. However, the current approach used for writing patterns has some limitations: for example, when a pattern (e.g., a sequence of elements with some specific characteristics) is satisfied by different sets of elements in the input stream, consumers do not have any operator for deciding which ones to *select*. This is a well known issue in the community working on CEP, known as the lack of customizable selection policies [1,94].

As far as implementation is concerned, EP-SPARQL queries are translated in logic expressions in the ETALIS Language for Events (ELE) [95] and computed at run-time using the event-based backward chaining algorithm of the ETALIS [96] engine. ETALIS converts queries to Prolog rules and evaluates them in a single thread of execution; parallel and distributed evaluation is currently not supported.

The current evaluation of the approach is based on synthetic workloads of relatively small scale (up to 50,000 triples). The query processing engine evaluates both background and streaming data in a single component, which also supports stream reasoning. In their experiments, the authors show the benefits of their implementation by comparing it with Esper, which is a well known commercial CEP engine [97]. If we look at the results, then the cost of reasoning becomes immediately evident: even when considering a simple workload (subclass inference), reasoning introduces a high processing latency, in the order of seconds. Additional experiments, including a comparison with C-SPARQL and CQELS, are described in [77]. According to this work, EP-SPARQL exhibits better processing time and scalability than C-SPARQL in various situations (with speedups of up to three order of magnitude), while it cannot achieve the same level of performance of CQELS.

As a final comment, it is worth mentioning that the authors of EP-SPARQL explored the integration of streaming information and historical data by implementing in ETALIS the capability to store and retrieve time-annotated data [98].

Answer set programming. Another proposal for using logic programming, and in particular Answer Set Programming (ASP), is described in [73]. The authors propose an extension to Answer Set Programming to deal with dynamic data. Although the authors mention an implementation based on a ASP solver, no evaluation is currently described for the proposed approach.

A similar approach is proposed in [99], where the authors present the StreamRule framework, which combines an engine for stream processing and filtering (implemented using CQELS) and a non-monotonic rule engine for ASP. Despite some preliminary investigations, no detailed evaluation is currently available to assess the performance of the StreamRule framework.

4.2.3. Time-aware reasoning

TA-SPARQL. TA-SPARQL (Time Annotated SPARQL) [74] is presented as a semantic stream manager system explicitly conceived to store and query time-annotated RDF data coming from sensors. The authors first introduce an extended version of RDF, similar to the one adopted by C-SPARQL, where resources are annotated with timestamps. Then, they introduce the TA-SPARQL query language, which extends SPARQL to support queries that refer to the past. Currently, the proposed model only allows one-time queries. Although the authors recognize the importance of continuous queries in streaming scenarios, they do not currently support them in their language. The model has been implemented in a prototype system running on top of the Tupelo semantic content manager, as part of a project that aims at storing and processing data coming from NEXRAD radars [100]. Currently, no experimental evaluation of the performance of the system is publicly available.

Temporal RDF. A similar approach is presented in [75], where the author present an extension to RDF to capture the notion of time, enabling reasoning over time enriched RDF data. The authors provide a semantics for temporal RDF graphs and show how reasoning over temporal RDF does not add extra asymptotic complexity with respect to non-temporal RDF.

Similar approaches for including time into RDF, as well as proposals for time and context-aware query languages, have been explored in [101–107]. All these solutions lack the support for continuous queries.

4.2.4. Surveys and visions

The need for more complex forms of stream reasoning has been identified in several papers [108,6,7]. These papers also highlight some of the main challenges that still need to be addressed to enable stream reasoning: provide a good model that combines evolving data representation and real-time evaluation, including pattern detection; extend existing reasoning algorithms to support continuous processing, possibly through incremental evaluation; exploit parallel and distributed computation to enable reasoning to scale to larger amounts of data. From a theoretical viewpoint, a few work focused on defining new logics for reasoning with streaming time-annotated data. An extensive review and comparison of these proposals can be found in [109].

As already mentioned while describing the systems, some research efforts were devoted to investigate the methodologies for measuring and comparing the performance of stream reasoning systems. This includes theoretical work [76], as well as concrete proposals for benchmarks [110,77–79] with concrete measurements of existing solutions.

Finally, an interesting contribution comes from [111], a vision paper in which the authors study stream reasoning by focusing on the relations between the reasoning task and the existence of ordering among information items. In particular, they investigate how the processing algorithms are influenced by the presence of a natural order (e.g., order of arrival, or timestamp order) and by the request for some application-specific order (e.g., top-k reasoning). The authors study how existing technologies support or exploit ordering properties. In doing this, they highlight open challenges and present possible research directions.

5. Survey: discussion

In the previous two sections, we surveyed existing research efforts in the area of stream reasoning and in related fields. Starting from our analysis, and in comparison with the requirements identified in Section 2, we can make a few important considerations on the current state of the field.

Background work. First of all, we notice that the solutions developed in the areas of reasoning and stream processing cannot fully satisfy all the requirements identified in Section 3.

Stream processing systems introduce a new interaction model, based on continuous queries, which are installed in the system and get repeatedly evaluated as new data is received. This enables on-the-fly processing of streaming information. However, in most cases, traditional stream processing engines focus on a reduced number of operators [39,41,36] and trade expressiveness for efficiency. This badly matches with the requirements for integration and expressivity identified in Section 2.

Additionally, only some stream processing systems, mainly in the domain of DSMSs, enable the interaction with background data stores, and only a few solutions offer operators to store and retrieve historical data coming from the stream. Both the integration of streaming and background data and the management of historical data were identified as key requirements in Section 2.

Finally, only some systems, typically in the domain of CEP, provide explicit operators for capturing temporal patterns over streaming information [1,33]. Even though DSMSs allow the access the timestamp of data items, they limit the scope of processing to the current evaluation window [31], and this reduces the possibilities for effectively managing time during processing, which was identified as a key requirement.

At the other side of the spectrum there are reasoning systems developed in the context of Semantic Web. The usage of semantic annotations and ontologies favors the integration of multiple data sources, while reasoning allows the derivation of implicit knowledge from explicit information.

However, traditional reasoning systems consider information as a set of statements, which describe the domain of interest and do not change (or rarely change) over time. Because of this, they cannot satisfy the requirement for integration of streaming (dynamic) data, as identified in our motivating scenarios. Similarly, they do not offer support for time management in their data and processing model, and they do not consider historical data.

Stream reasoning. Stream reasoning systems have been developed to combine the benefits of stream processing and reasoning. We can extract some interesting observation starting from the description in Section 4 and from the classification in Table 2.

First of all, almost all the systems developed in the area of stream reasoning inherit the processing and interaction model of DSMSs, which is based on continuous queries. In this processing model, the streaming data is partitioned using windows and each query evaluation is performed within the boundaries of the current window. We classified them as *semantic stream processing* systems. The only exceptions are represented by EP-SPARQL and Answer Set Programming (classified as *semantic event processing* system), in which pattern detection is used as the primary processing mechanism. As a consequence, these are also the only proposals to include explicit temporal operators.

This processing model makes it easy to integrate background data about the domain of analysis. In fact, almost all the system that we analyzed enable integration of stream and background data during processing.

All the considered systems extend RDF with temporal annotations. In most of them, time is encoded as a single timestamp, but there are also a few cases in which time intervals have been adopted.

Since the reasoning process is computationally expensive, existing solutions implement only reduced reasoning capabilities, if any. Some of them focus on single properties, while other consider subsets of RDFS or OWL2 RL and incremental reasoning is applied only under some restrictive assumptions on the adopted data and processing models.

Despite some promising investigation (e.g., EP-SPARQL), currently no solution fully integrates both stream processing capabilities and historical data management. Some systems offer on-the-fly processing using continuous queries. Other systems are designed to store time-annotated data and provide operators for querying and retrieving it. None of them, however, offer a unifying approach for both kinds of processing.

As we observed in our analysis, existing stream processing systems have been tested and evaluated on relatively small scenarios. However, the capability of managing large volumes of data is a key requirement in many application scenarios, as identified in Section 2. The recent research on reasoning for Semantic Web [112,113,46] has demonstrated that some problems appear only when increasing the scale of the problem under analysis. Accordingly, validating an algorithm over small scenarios may not be sufficient to demonstrate its general applicability and scalability.

Along the same line, only two systems currently implement parallel and distributed processing. This aspect is of primary importance, since the resources of a single machine are limited and may easily become insufficient as the scale grows.

More in general, as observed in recent papers [76–79], assessing and comparing the processing models and performance of stream reasoning solutions is a problem per-se, which still requires further investigations.

Interestingly, no existing system addresses the problem of uncertainty. As previously discussed, uncertainty management represents an important issue in streaming applications. Similarly, no existing system currently includes customizable QoS policies.

Finally, in Section 2 we identified the management of distributed scenarios as a key requirement for stream reasoning, and we highlighted the challenges that it introduces. These challenges, however, have not been considered so far in existing systems. In the following section, we start from the present analysis of open challenges in the field to propose a research agenda for addressing them.

6. Next steps: a research agenda

In the previous sections, we started from the analysis of requirements for stream reasoning systems (Section 2) and from the survey of existing systems (Sections 3 and 4), to identify the open challenges in the area of stream reasoning (Section 5).

This section starts from these premises and presents a research agenda with a concrete description of some steps required to drive the design and implementation of future stream reasoning systems. We believe that future research should target both the theoretical foundations, algorithms, techniques, and implementation of stream reasoning that could enable building efficient and scalable tools. Because of this, we organize our discussion as follows. First, we focus on *system models* for representing *data* and *operations* on data (Section 6.1). Second, we consider aspects related to the *system implementation* (Section 6.2). Finally, we discuss the problems that derive from the application and evaluation of the solutions and systems in the area of stream reasoning (Section 6.3).

In our analysis, we start from the main challenges and open issues and we present research directions and possible solutions proposed in related fields. Fig. 3 summarizes our research agenda, while Table 3 shows how the topics covered by the research agenda map to the requirements of our use-cases.

Table 3
Relation between requirements and research agenda.

Scenario	Integration		Time manag.		Distribution		Big data manag.	Efficiency		Expressivity			Uncertainty manag.	Historical data	Quality of service
	Different sources	Streaming/background data	Data model	Processing model	Distrib. sources	Distrib. processing		High throughput	Low latency	Reasoning	Temporal operators	Data transformation			
Time model		✓	✓	✓	✓			✓	✓		✓			✓	
Historical data model														✓	
Uncertainty model	✓												✓		
A model for Q&R		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓		✓	
Uncertainty prop. model													✓		
Information management					✓	✓	✓	✓	✓					✓	
Reasoning on big data						✓	✓	✓	✓	✓					
Incremental reasoning							✓	✓	✓	✓					
Approximate reasoning							✓	✓	✓	✓					✓
Efficient query evaluation						✓	✓	✓	✓						
Management of bursts						✓	✓	✓	✓						✓
Operator placement					✓	✓	✓	✓	✓						

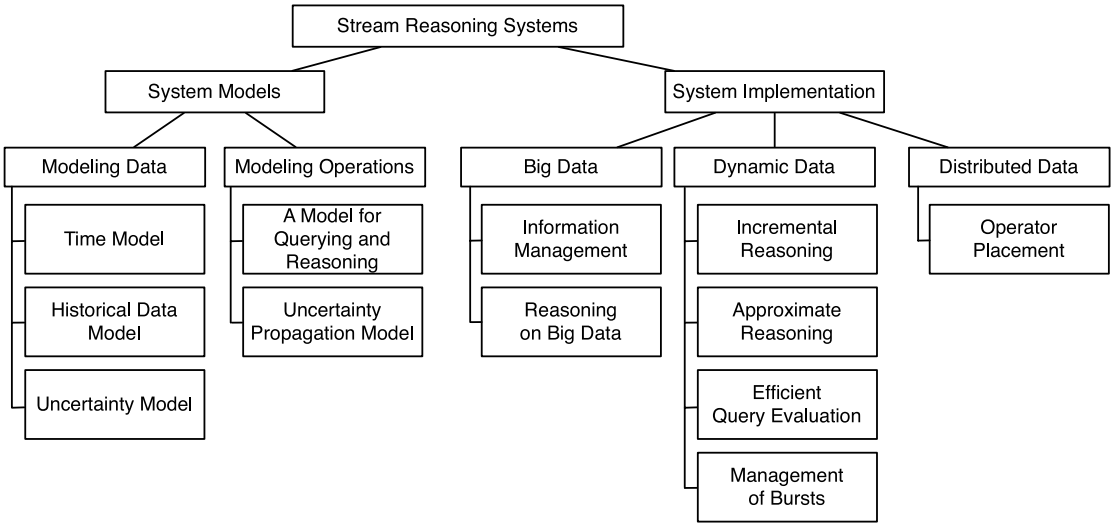

Fig. 3. A research agenda for stream reasoning.

6.1. System models

This section focuses on the models for representing, processing, and retrieving information in stream reasoning systems. As mentioned in Section 5, the definition of such models vastly remains an open research issue.

6.1.1. Modeling data

RDF has been widely accepted as the data model for representing semantically annotated information. While it is well suited for static knowledge, it needs to be extended to consider data that changes over time.[5] The research challenges in this context involve the definition of a model for *time* and for time-changing data. Additionally, in many application scenarios for stream reasoning, information comes with some degree of *uncertainty*. Understanding and modeling such uncertainty is critical for an efficient and correct management of dynamic knowledge.

Time model. Given the importance of dynamic and time-changing data in streaming applications, we see the definition of a suitable model for time as one of the first and most critical aspects for building a solid theoretical foundation for stream reasoning. As we show in Table 3, a proper choice of the time model is necessary to satisfy the requirements connected with the management and processing of dynamic data, including the integration of streaming and background data, the management of historical data, and the capability to perform reasoning on time-changing data. Moreover, the semantics of time determines how we can approach the synchronization issues that traditionally arise in distributed settings.

In the past, several models have been proposed for annotating information with time. Most stream processing systems extend the data model by adding a single timestamp to each data item. As shown in Section 4, this approach is also adopted by most of the existing proposals in the area of stream reasoning. Intuitively, the single timestamp model is appropriate for expressing the occurrence of events. However, when considering facts, or states in the domain of analysis, other representations of time may be more suitable. As shown in Section 4, some systems exploit an interval based representation, where two timestamps are used, indicating the lower and upper bounds of the interval in which a piece of information is considered as valid.

The introduction of interval representations dates back to temporal databases [54] and was adopted in stream processing solutions both in the DSMSs domain [114] and in the CEP domain [41,36]. The use of time intervals enables the definition of a rich set of temporal relations. For example, an interval I_1 can follow and interval I_2, start, or finish together with I_2, overlap, or include I_2. An extensive study of the relations between time intervals is present in the pioneering work of Allen [115]. As discussed in [116], different semantics can be provided to define the temporal relations between time intervals (e.g., to define the immediate successor of an item), each of them satisfying different properties (e.g., associativity).

Finally, as mentioned in Section 2, the use of an ontology for time [9] has been proposed as part of the Semantic Sensor Web project. This ontology defines the main concepts related with time (e.g., time instant, interval, duration) and the main relations between these concepts (e.g., equality, overlapping, ordering). From a theoretical viewpoint, the ontology provides the same model and level of expressiveness as the interval representation described above. However, it provides flexibility and simplifies the integration among different time notations by introducing a precise vocabulary that includes dates, timezones, and temporal unit types.

Beside introducing a representation of time, the time model needs to specify its *semantics*. In particular, there are two common ways in which timestamps may be assigned to data items [117,1]: based on the time when they enter the processing system, or based on some application time (e.g., the time when they are generated). In the former case, it is easy to evaluate time relationships, since the processing system is guaranteed to receive items in timestamp order. The latter case presents more difficulties, since the information items can be received out of order due to unsynchronized application clocks at sources [118], network latencies, non-order-preserving or lossy communication channels. A detailed study, and a proposal to cope with out-of-order using heartbeats is presented in [117].

As we have seen in our analysis in Section 4, all existing proposals in the area of stream reasoning consider an application time in which the timestamp represents the time when a data item is generated at its source and it is assumed that the data enters in the processing system in timestamp order. Finding the best time model for stream reasoning is an open research question. We identify here three quality metrics that can guide the choice. (i) Expressiveness: the time model should enable expressing all the timing and ordering relations among information items required by applications. (ii) Simplicity: the temporal model should provide

[5] An additional challenge, beyond the scope of this paper, involves the integration with existing (static and dynamic) non-RDF data.

an intuitive semantics, making it easy to express and understand temporal relations. (iii) Efficiency: evaluating temporal relations should not introduce unacceptable overheads.

The importance of defining a data and time model for stream reasoning has been recognized by various research communities. Recently, a new W3C Community Group on RDF Stream Processing (W3C RSP) [119] has started with the declared goal of defining a common model for producing, transmitting, and continuously querying RDF streams.

Historical data model. In Section 2, we identified the capability of providing access to historical data as a requirement for stream reasoning tools. This demands for data and storage models that keep track of information changes and offer suitable mechanisms for querying and retrieving older data.

Defining these models implies answering several research questions that are strictly related to the representation of time-annotated data: how should the validity time of information be represented? What is the semantics of new information items? Do they complement previous knowledge or update/invalidate (part of) it? Is it possible to store only partial (aggregated) information about the past? Who defines which information needs to be preserved and how?

As discussed in Section 4, several approaches have been proposed in the field of time-aware reasoning that provide answers to some of the questions above, introducing theoretical models for representing time changing information and reasoning about it. Moreover, solutions for integrating stream processing with historical data have been designed in the past by the database community, and implemented in some existing DSMSs [120,121]. Finally, models and techniques for compressing historical data with minimal information loss have been also investigated in [122].

As observed in Section 4, no existing solutions in the area of stream reasoning integrates both management of streaming data and flexible management of historical data. Future research in this area should target (i) the definition of a unified model for time-annotated data that spans both streaming and historical data, and (ii) the specification of operators and abstractions for storing and retrieving (portions of) the streaming data.

Uncertainty model. In Section 2, we identified the ability to deal with uncertainty as a requirement of many application scenarios. We distinguish between the uncertainty associated to input data and the uncertainty associated to the processing step [123]. More in particular, input data can be: (i) imprecise, (ii) incomplete, (iii) incorrect or inconsistent. For instance, in sensor networks, imprecision may derive from the limited resolution of sensors, while incompleteness may be caused by data loss for temporary lack of communication or battery issues. Incorrectness or inconsistencies are frequent in application that involve Web data. Moreover, the processing can introduce some degree of approximation. Additionally, an incomplete or inaccurate modeling of the domain of analysis can lead to produce incorrect or imprecise outputs [123].

Several ways for modeling uncertainty have been studied in the past [124]. They include probability theory, Bayesian networks, fuzzy logics, and many other formalisms. The recent literature also presents proposals in the area of stream processing [125–128,123].

Most of these solutions rely on an explicit encoding and representation of uncertainty inside data items. For instance, [123] adopts random variables to represent received information, thus making it possible to encode measurements errors in sensor applications. We believe that these approaches can be complemented with techniques developed in the area of Semantic Web and reasoning for detecting and solving data inconsistencies [129], thus contributing in satisfying the requirement for integration, as shown in Table 3.

There are various metrics for evaluating a model for uncertainty: beside expressiveness, flexibility, and precision, simplicity is also very relevant. End consumers may be interested in receiving some precise indication about the level of certainty associated to the results they get. However, this should not negatively impact the compactness and readability of information.

6.1.2. Modeling operations

After analyzing the models for data representation, we now focus on the models for representing operations on data. In particular, we focus on the models for reasoning and querying dynamic data and present these models together since, as we will better motivate in the following, we consider them as strictly integrated. Because of that, we claim that both querying and reasoning can be combined into a single unifying processing model for stream reasoning.

A model for querying and reasoning. The query and reasoning models determine the expressive power of the system and the amount of computation it needs to perform. The query model represents the interface for gathering information from the system, while reasoning can be used to enrich query results.

In the domain of stream processing, DSMSs inherit their query model from relational databases, with additional windowing constructs for selecting the portions of input data to consider. They do not *derive* new information, but rather *transform* (e.g., select, join, aggregate) input data to present it as required by the consumers and they do not include any form of reasoning. As shown in Section 4, most of the existing proposals for stream reasoning inherit the query model of DSMSs. Most importantly, they preserve the traditional model of reasoning for static data, which ignores time information associated to data. As already observed in previous work [6], this model introduces some issues related to the semantics of processing: since a stream is observed only through a window of a finite (and often pre-determined) size, it is possible that the processing does not consider the entire input and therefore is incomplete.

On the other hand, CEP systems target at *defining* new, implicit, knowledge (in the form of composite events) starting from time-based patterns of primitive events that can be directly observed from the external environment. Because of this, we see a significant common ground between reasoning tools and CEP systems. Although they move from different perspectives and adopt different formalisms, both of them aim at extracting and presenting new, implicit knowledge. Reasoning tools consider static data and (potentially) complex forms of reasoning that do not involve time relations between facts. CEP systems consider relatively simple form of processing (typically pattern matching) over dynamic data. Time is treated as a first-class citizen and the time relationships between data elements are widely considered.

For this reason, we claim that future research could investigate the feasibility and benefits of a unifying query model and semantics that captures both worlds. Some preliminary work already goes in the same direction [72]. Interesting areas of investigation include the use of temporal logics, which have been proposed in the past for reasoning over time annotated knowledge [130–133]. Temporal logics have also been used by some CEP systems to formally define the semantics of their operators [33]. A unified model based on logic would enable a precise definition of soundness and completeness, overcoming the issues deriving from the usage of windowing mechanisms.

Finally, as we have seen in Section 4, some proposals have started investigating the use of inductive and/or statistical reasoning [66]. Introducing processing abstractions for statistical reasoning could enable the inference of new rules from the observed data and potentially improve the results provided to the consumers, and attempts to unify logical and statistical reasoning have already been discussed in the literature [134].

As we show in Table 3, the choice of a suitable processing model affects almost all the features required by our use-cases. Most importantly, it balances between the expressiveness offered to consumers and the processing performance, efficiency, and scalability.

Uncertainty propagation model. In Section 6.1.1, we discussed the importance of identifying suitable models for representing uncertainty. After such models have been defined, the system needs to take into account uncertainty during the processing. This makes the end consumers aware of the degree of certainty associated to the results they obtain. Depending on the adopted models and techniques, the evaluation and propagation of uncertainty may impact the performance. For instance, taking into account potential measurement errors and incorporating them into the processing may require using expensive statistical computation.

Efficiency in presence of uncertainty has been a main concern in stream processing systems and led to the design of promising techniques [125–128,123]. Finding a good balance between the advantages of a model in terms of accuracy, and simplicity, and the costs required for managing it constitutes a major area of investigation for future research.

6.2. System implementation

Our analysis of application scenarios highlighted some key scalability requirements for stream reasoning tools. An effective solution for stream reasoning should be able to consider big data (Section 6.2.1), which is dynamic and potentially updated at high frequency (Section 6.2.2), and generated from a large number of (geographically distributed) sources of information (Section 6.2.3). In the following, we focus on these three aspects. For each of them, we present a number of implementation and engineering issues and describe mechanisms, techniques, and solutions for them that were developed in related fields.

6.2.1. Big data

We consider two main challenges related to big data: information *management* and information *processing* (in particular, reasoning in presence of big data).

Information management. This section presents the challenges related to information management, query, and retrieval in presence of big data.

Since stream reasoning systems may need to access large volumes of data during processing, it is crucial to enable fast mechanisms for information retrieval to reduce the response time of the system. To this purpose indexed data structures, as well as caching techniques, can be properly exploited. Several optimization techniques for database systems have been introduced in the literature (e.g., size of different tables [135]) that adapt to the properties of stored data. Furthermore, ad-hoc algorithms could analyze the patterns of access to information to determine which data is more relevant for consumers and consequently optimize its retrieval.

Other challenges arise in presence of data distributed over multiple nodes. In this situation, it becomes important to consider the physical location of information to minimize the delay for data retrieval. This demands for communication protocols that define where the data is stored and how it is routed to interested recipients. The research in this area may benefit from previous results from distributed information retrieval [136], content delivery networks [137], and protocol for content-based communication [138]. Further investigations on the issue of processing distributed data are reported later on, in Section 6.2.3.

Reasoning on big data. Reasoning over large volumes of data constitutes a challenging task. Currently, several parallel and distributed have been proposed in literature [112,113,46,139,47,50,140–144]. Some of these techniques have shown RDFS and OWL entailment over billions of RDF triples with a few hours of processing on clusters with few dozens of machines. These works constitute a valuable starting point for future research, not only for their technical merits, but also because they highlight some key issues that appear at large scale, and propose solutions or general principles for overcoming them. For example, they investigate how to split data over multiple machines, or processing cores, which data structures to adopt, and how to better exploit the limited size of the main memory, how to reduce the expensive communication between processing nodes.

Future work should be able to extend these solutions and enable reasoning over data that changes frequently. Some preliminary investigations in this direction already exist [67]; they complement the techniques designed for reasoning over dynamic data presented in the following section.

6.2.2. Dynamic data

From the perspective of reasoning, dynamic data requires a constant re-evaluation of inferred information. Due to the complexity of such operation, techniques for *incremental* and *approximate* reasoning have been proposed. At the same time, the presence of continuous updates requires *low delay* evaluation of continuous queries, even in presence of information *bursts*.

Incremental reasoning. Performing RDFS reasoning to compute the full materialization of relatively small datasets (a few millions of triples) may require several minutes on large clusters [112]. In presence of frequently changing data and time constraints, it is not possible to repeatedly apply traditional reasoning algorithms over the entire knowledge base and this demands for incremental techniques that only consider data that is influenced by changes.

Research in this area is of primary importance, since it aims at reducing the gap between the frequency of changes that characterizes many application domains and the amount of time demanded by complex reasoning techniques. As Table 3 shows, this impacts both expressiveness and efficiency.

The problem of updating derived information upon changes in the knowledge base has been widely studied by the database community in the context of view maintenance and deductive databases. More in particular, the idea of incrementally updating derived information has been studied since the beginning of the 1980s [145], leading to two main algorithms: DRed (Delete and Rederive) [146], and PF (Propagate Filter) [147]. These works have demonstrated how one of the most crucial aspects related to incremental reasoning is the identification and removal of derivations that are no longer valid when information changes. Indeed, this might require to keep track of some or all the dependencies between explicit and derived knowledge. To this respect, both algorithms share the same idea: when some information is removed, they first compute an overestimation of the derived knowledge that needs to be deleted, and then re-derive the information that is still valid. Extensions to these approaches have been proposed in the last few years [83]. A parallel and distributed implementation based on these algorithms has been recently proposed and evaluated on a subset of the RDFS entailment [148], even though it does not consider time annotated data. Furthermore, there exist solutions based on very specific languages and restricting assumptions. This is the case for the technique developed for C-SPARQL queries [62]: it targets the reasoning performed to enrich query results and exploits some intrinsic properties of (a subset of) C-SPARQL to simplify the processing.

A precise analysis of the features and operators used for reasoning could help understanding their complexity, thus making it easier to decide what type of reasoning is compatible with the processing of streams. One possible way to reduce the cost of reasoning over streams is to limit amount of knowledge that is being

materialized. While in static scenarios a complete materialization of all possible derivations can be desirable, different approaches may be better suited for streaming applications. We identify the analysis of the best balance between pre-computation (and maintenance) of the derived information, and on-demand computation as another key topic that should drive the research in the area.

Approximate reasoning. Even with efficient incremental algorithms, reasoning may still be too expensive to be performed on-the-fly on streaming data but it may be acceptable in some contexts to trade completeness or precision of reasoning for response time. As shown in Table 3, this targets the requirement for a user-defined quality of service.

Solutions for approximate reasoning have been proposed in the literature [149], and also adapted to streaming scenarios [64,65]. Nevertheless, approximate reasoning vastly remains an open problem. In this context, we foresee two research directions: (i) performance improvement of existing solutions, and (ii) cost prediction, to model the computational costs connected to a certain reasoning task, to decide whether approximate methods should be applied.

Efficient query evaluation. Stream reasoning systems need to update the answers to continuous queries as new information is received. Dealing with high frequency streams of data is challenging, since it introduces some hard constraints on the processing time. More in particular, we identified in Section 2 two requirements for efficiency: high throughput and low response latency. On the one hand, high throughput enables the system to scale on the amount of input data that it can handle in a given period of time. On the other hand, low response latency enables the generation of results in a timely way, which may be critical in some application scenarios (e.g., emergency management).

Both metrics require efficient evaluation of input information. This has been widely investigated in stream processing systems [37,34,41,36], proposing techniques for rewriting the deployed queries to optimize their execution. These solutions focused on re-defining the structure of single queries, but also on sharing operators between multiple queries, as proposed in the field of multi-query optimization [150].

Other works investigated the benefits of parallel processing to reduce response time using not only multi-core CPUs, but also co-processors like GPUs [151] or FPGAs [152,153]. Investigating how these processing technologies could be adopted in the domain of stream reasoning certainly represents an important research direction.

Future research also needs to consider how query evaluation can be combined with the reasoning process. Some traditional knowledge bases pre-materialize all the (implicit) knowledge that can be inferred from (explicitly) stored information to speed up query answering. In streaming systems, the set of deployed queries can limit the scope of reasoning for extracting only required information.

Management of bursts. Strictly connected with on-the-fly processing of data streams produced at high rate is the management of bursts.

In the area of stream processing, the problem has been studied primarily by the community working on DSMSs, which developed several load shedding techniques [154–159]. Load shedding aims at cutting out parts of the input streams when they saturate the computational capacity of the system. Existing proposals try to identify which data items have less probability to participate in the final results, such that removing them has only a marginal impact on the computation of output.

Another approach for managing bursts is represented by elastic stream processing [160–162], which dynamically adjust the number of computational resources adopted based on the current load of the system. This approach is particularly suitable for deployments on computational cluster or clouds.

6.2.3. Distributed data

Settings that require distributed information management are common. This incurs additional problems which are well researched in distributed information systems. Data locality is a general principle to improve the performance if the computational resources are placed at different locations. As an example, processing of financial information for algorithmic trading often takes place close to the location when data is generated, to reduce the overall latency [163]. Data and processing can be clustered to limit as much as possible expensive inter-node communication.

Operator placement. The problem of how to distribute the computation is well known in the context of stream processing and often referred to as the *operator placement* problem. Solutions for this problem aim at finding the best allocation of processing tasks over the computational resources, while possibly taking into account the characteristics of resources (e.g., computational, memory, storage or communication capabilities, availability of specific hardware, etc.) and additional domain specific constraints. Existing approaches are designed for continuous queries over streams of data: they break each query into its basic constituent operators and then decide where to deploy them. This enables for both concurrent evaluation of different queries at different nodes, as well as for incremental evaluation of a single query over multiple machines. Incremental evaluation is particularly important since it enables significant optimizations, like pushing the operators that filter data as close as possible to the sources of information.

Several solutions have been presented for operator placement, each one targeting a different scenario (e.g., large scale distributed systems vs. local area network or clusters) and focusing on different goals (e.g., load balancing vs. minimization of delay or bandwidth usage) [164–167,35,28,168–174]. In the context of stream reasoning, these works present one significant limitation that needs to be addressed in future research: they only focus on streaming data and do not consider the presence of stored data or background knowledge.

Beside the problem of operator placement, some previous work has defined and analyzed communication protocols between processing nodes that further optimize the way information is transferred. In this case, some solutions propose to create temporary buffers at intermediate nodes and to store information there instead of transmitting it to the final recipients. Information can be pulled from these buffers only when (and if) it becomes relevant for processing [175], thus reducing the overall network traffic. Some solutions push this idea to its limits by storing some information directly at the sources [176]. These proposals are valuable in the context of stream reasoning, since they can suggest effective ways for combining streaming and stored data.

6.3. Stream reasoning in action

The application scenarios for stream reasoning are heterogeneous and stress different requirements. For example, certain scenarios deal with frequently changing data and target low response times, but do not demand for expressive reasoning. Others can sacrifice the response time and the processing efficiency to increase the expressiveness of the computation.

This drives to a critical research question. Is it possible to provide a single solution that is suitable for all applications? If not, which are the best models, abstractions, and implementation mechanisms to address specific parts of the problem space?

On the one hand, answering this question demands for a more precise analysis of the application requirements. Some existing works have been tested in real or realistic scenarios. However, it is still not clear if different models for data representation and processing, or more complex forms of reasoning can provide additional benefits to the applications and, if so, to what extent. On

the other hand, most of the proposed solutions are still in the form of research prototypes, and often introduce significant assumptions and limitations. For example, most of them have been tested on relatively small scenarios, often assuming that the knowledge base can fit into the main memory. Similarly, key techniques for dynamic data management, like incremental reasoning, have not been tested on real scenarios. In this context, it becomes difficult to quantify the impact of each design choice. For instance, it is difficult to measure how the volume of data impacts on the response time of the system, or which rate of information update is supported with a given reasoning model and complexity.

Next to the problem of understanding the best abstractions and design choices for some specific scenarios, it is critical to assess and compare concrete systems, to measure their benefits and limitations. Initial work in the area has demonstrated the complexity of this task, even when considering systems with similar features [78,77,79]. As recognized in some initial work [76], future research in this area should target at (i) identifying the critical Key Performance Indicators (KPIs) of the systems, (ii) understand which are the parameters that mostly influence them, and (iii) devise stress tests for controlling these parameters and to measure their impact.

7. Conclusions

In this paper, we provided a detailed analysis of stream reasoning, a new and vastly unexplored research area. We first looked at some concrete application scenarios to extract the requirements for stream reasoning. Then, we analyzed existing proposals in the area, discussed their properties, and highlighted their advantages and limitations. Starting from this analysis, we isolated some key challenges that need to be addressed to offer full fledged tools for stream reasoning.

We moved from the current state of the stream reasoning to propose a research agenda to further advance in this field. We believe that future research should look both at defining the theoretical foundations for stream reasoning and at designing algorithms and tools for a scalable and efficient processing.

From a theoretical point of view, suitable models for data representation and processing are still missing. To satisfy the requirements of current dynamic scenarios, time plays a central role in the definition of such models. Therefore, they must enable seamless integration of streaming, background and historical data, define and identify complex temporal patterns, and reason over information that changes over time.

From an implementation perspective, there is the need for strong methods and tools that support the theoretical abstractions. The complexity of the problem demands for advanced algorithms for data processing, efficient communication protocols, and solid implementations that best exploit available resources.

To conclude, even though stream reasoning is not yet a mature field of research, this survey illustrates how the research community is currently addressing this set of problems with new principles, algorithms, and solutions to advance the state of the art in this new and challenging research domain.

Acknowledgment

This research has been funded by the Dutch national program COMMIT.

References

[1] G. Cugola, A. Margara, Processing flows of information: from data stream to complex event processing, ACM Comput. Surv. 44 (3) (2012) 15:1–15:62.
[2] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '02, ACM, New York, NY, USA, 2002, pp. 1–16.
[3] D.C. Luckham, The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
[4] O. Etzion, P. Niblett, Event Processing in Action, first ed., Manning Publications Co., Greenwich, CT, USA, 2010.
[5] OWL Working Group, OWL 2 web ontology language document overview, 2009. URL: http://www.w3.org/TR/2009/REC-owl2-overview-20091027/.
[6] E. Della Valle, S. Ceri, F. van Harmelen, D. Fensel, It's a streaming world! reasoning upon rapidly changing information, IEEE Intell. Syst. 24 (6) (2009) 83–89.
[7] E. Della Valle, S. Ceri, D. Barbieri, D. Braga, A. Campi, A first step towards stream reasoning, in: J. Domingue, D. Fensel, P. Traverso (Eds.), Future Internet—FIS 2008, in: Lecture Notes in Computer Science, vol. 5468, Springer, Berlin, Heidelberg, 2009, pp. 72–81.
[8] A. Sheth, C. Henson, S. Sahoo, Semantic sensor web, IEEE Internet Comput. 12 (4) (2008) 78–83.
[9] Time ontology in OWL, Time ontology in OWL, 2012. URL http://www.w3.org/TR/owl-time/.
[10] M. Compton, P.M. Barnaghi, L. Bermudez, R. Garcia-Castro, Ó Corcho, S. Cox, J. Graybeal, M. Hauswirth, C.A. Henson, A. Herzog, V.A. Huang, K. Janowicz, W.D. Kelsey, D.L. Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K.R. Page, A. Passant, A.P. Sheth, K. Taylor, The ssn ontology of the W3C semantic sensor network incubator group, J. Web Semant. 17 (2012) 25–32.
[11] L. Atzori, A. Iera, G. Morabito, The internet of things: a survey, Comput. Netw. 54 (15) (2010) 2787–2805.
[12] F. Lecue, S. Kotoulas, P.M. Aonghusa, Capturing the pulse of cities: opportunity and research challenges for robust stream data reasoning, in: AAAI Workshops, 2012.
[13] S. Tallevi-Diotallevi, S. Kotoulas, L. Foschini, F. Lécué, A. Corradi, Real-time Urban monitoring in dublin using semantic and stream technologies, in: H. Alani, L. Kagal, A. Fokoue, P. Groth, C. Biemann, J. Parreira, L. Aroyo, N. Noy, C. Welty, K. Janowicz (Eds.), The Semantic Web ISWC 2013, in: Lecture Notes in Computer Science, vol. 8219, Springer, Berlin, Heidelberg, 2013, pp. 178–194. URL: http://dx.doi.org/10.1007/978-3-642-41338-4_12.
[14] A. Wagner, S. Speiser, A. Harth, Semantic web technologies for a smart energy grid: Requirements and challenges, in: Proceedings of 9th International Semantic Web Conference, ISWC2010, 2010, pp. 33–37.
[15] R. Shojanoori, R. Juric, Semantic remote patient monitoring system, in: Telemedicine and e-Health, 2013.
[16] F. Paganelli, D. Giuli, An ontology-based context model for home health monitoring and alerting in chronic patient care networks, in: 21st International Conference on Advanced Information Networking and Applications Workshops, 2007, AINAW '07, vol. 2. 2007, pp. 838–845.
[17] P. Groth, A. Gibson, J. Velterop, The anatomy of a nanopublication, Inf. Serv. Use 30 (1–2) (2010) 51–56.
[18] B. Mons, J. Velterop, Nano-Publication in the e-Science Era, 2009.
[19] A.J. Williams, L. Harland, P. Groth, S. Pettifer, C. Chichester, E.L. Willighagen, C.T. Evelo, N. Blomberg, G. Ecker, C. Goble, B. Mons, Open phacts: semantic interoperability for drug discovery, Drug Discovery Today 17 (2122) (2012) 1188–1198. URL: http://www.sciencedirect.com/science/article/pii/S1359644612001936.
[20] A.J. Gray, P. Groth, A. Loizou, S. Askjaer, C. Brenninkmeijer, K. Burger, C. Chichester, C.T. Evelo, C. Goble, L. Harland, et al. Applying linked data approaches to pharmacology: architectural decisions and implementation, Semantic Web, 2012.
[21] W. van Hage, V. Malais, G. de Vries, G. Schreiber, M. van Someren, Abstracting and reasoning over ship trajectories and web data with the simple event model (sem), Multimedia Tools Appl. 57 (2012) 175–197.
[22] P. Mika, Flink: semantic web technology for the extraction and analysis of social networks, Web Semant. Sci. Serv. Agents World Wide Web 3 (23) (2005) 211–223. URL: http://www.sciencedirect.com/science/article/pii/S1570826805000089.
[23] G. Erétéo, M. Buffa, F. Gandon, O. Corby, Analysis of a real online social network using semantic web frameworks, in: The Semantic Web-ISWC 2009, Springer, 2009, pp. 180–195.
[24] M. Balduini, E. Della Valle, D. DellAglio, M. Tsytsarau, T. Palpanas, C. Confalonieri, Social listening of city scale events using the streaming linked data framework, in: H. Alani, L. Kagal, A. Fokoue, P. Groth, C. Biemann, J. Parreira, L. Aroyo, N. Noy, C. Welty, K. Janowicz (Eds.), The Semantic Web ISWC 2013, in: Lecture Notes in Computer Science, vol. 8219, Springer, Berlin, Heidelberg, 2013, pp. 1–16. URL: http://dx.doi.org/10.1007/978-3-642-41338-4_1.
[25] R. Stephens, A survey of stream processing, Acta Inf. 34 (7) (1997) 491–541.
[26] M. Stonebraker, U. Çetintemel, S.B. Zdonik, The 8 requirements of real-time stream processing, SIGMOD Rec. 34 (4) (2005) 42–47.
[27] D. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, A. Maskey, A. Rasin, A. Singer, M. Stonebraker, N. Tatbul, Y. Xing, R. Yan, S. Zdonik, Aurora: a data stream management system, in: SIGMOD '03: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA, 2003, p. 666.
[28] D.J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A.S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, S.B. Zdonik, The design of the borealis stream processing engine, in: Second Biennial Conference on Innovative Data Systems Research, CIDR 2005, ACM, Asilomar, CA, USA, 2005, pp. 277–289.

[29] Y. Bai, H. Thakkar, H. Wang, C. Luo, C. Zaniolo, A data stream language and system designed for power and extensibility, in: CIKM '06: Proceedings of the 15th ACM International Conference on Information and Knowledge Management, ACM, New York, NY, USA, 2006, pp. 337–346.

[30] C. Cranor, T. Johnson, O. Spataschek, V. Shkapenyuk, Gigascope: a stream database for network applications, in: SIGMOD '03: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA, 2003, pp. 647–651.

[31] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, J. Widom, Stream: the stanford stream data manager, IEEE Data Eng. Bull. 26 (2003).

[32] A. Arasu, S. Babu, J. Widom, The CQL continuous query language: semantic foundations and query execution, VLDB J. 15 (2) (2006) 121–142.

[33] G. Cugola, A. Margara, Tesla: a formally defined event specification language, in: Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems, DEBS '10, ACM, New York, NY, USA, 2010, pp. 50–61.

[34] G. Cugola, A. Margara, Complex event processing with T-Rex, J. Syst. Softw. 85 (8) (2012) 1709–1728.

[35] G. Li, H.-A. Jacobsen, Composite subscriptions in content-based publish/subscribe systems, in: Middleware '05: Proceedings of the 6th ACM/IFIP/USENIX International Conference on Middleware, Springer-Verlag New York, Inc., 2005, pp. 249–269.

[36] N.P. Schultz-Møller, M. Migliavacca, P. Pietzuch, Distributed complex event processing with query rewriting, in: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, DEBS '09, ACM, New York, NY, USA, 2009, pp. 4:1–4:12.

[37] A. Adi, O. Etzion, Amit—the situation manager, VLDB J. 13 (2) (2004) 177–203.

[38] D.C. Luckham, J. Vera, An event-based architecture definition language, IEEE Trans. Softw. Eng. 21 (1995) 717–734.

[39] E. Wu, Y. Diao, S. Rizvi, High-performance complex event processing over streams, in: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, SIGMOD '06, ACM, New York, NY, USA, 2006, pp. 407–418.

[40] J. Agrawal, Y. Diao, D. Gyllstrom, N. Immerman, Efficient pattern matching over event streams, in: SIGMOD '08: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA, 2008, pp. 147–160.

[41] L. Brenna, A. Demers, J. Gehrke, M. Hong, J. Ossher, B. Panda, M. Riedewald, M. Thatte, W. White, Cayuga: a high-performance event processing engine, in: SIGMOD, ACM, New York, NY, USA, 2007, pp. 1100–1102.

[42] RDF, W3c recommendation: Rdf primer, 2012. URL: http://www.w3.org/TR/rdf-primer/.

[43] P. Hayes (Ed.) RDF Semantics. W3C Recommendation, 2004.

[44] H.J. ter Horst, Completeness, decidability and complexity of entailment for RDF schema and a semantic extension involving the OWL vocabulary 3, (2–3) 2005, pp. 79–115.

[45] OWL Working Group, OWL 2 web ontology language profiles, 2009. URL: http://www.w3.org/TR/2009/PR-owl2-profiles-20090922/.

[46] J. Urbani, S. Kotoulas, J. Maassen, F.V. Harmelen, H. Bal, Webpie: a web-scale parallel inference engine using mapreduce, Web Sem. Sci. Serv. Agents World Wide Web 10 (0) (2012) 59–75.

[47] J. Weaver, J. Hendler, Parallel materialization of the finite RDFS closure for hundreds of millions of triples, in: A. Bernstein, D. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, K. Thirunarayan (Eds.), The Semantic Web—ISWC 2009, in: Lecture Notes in Computer Science, vol. 5823, Springer, Berlin, Heidelberg, 2009, pp. 682–697.

[48] N. Heino, J.Z. Pan, Rdfs reasoning on massively parallel hardware, in: International Semantic Web Conference, vol. 1, 2012, pp. 133–148.

[49] S. Das, E.I. Chong, Z. Wu, M. Annamalai, J. Srinivasan, A scalable scheme for bulk loading large RDF graphs into oracle, in: ICDE, 2008, pp. 1297–1306.

[50] V. Kolovski, Z. Wu, G. Eadon, Optimizing enterprise-scale OWL 2 RL reasoning in a relational database system, in: P. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Pan, I. Horrocks, B. Glimm (Eds.), The Semantic Web ISWC 2010, in: Lecture Notes in Computer Science, vol. 6496, Springer, Berlin, Heidelberg, 2010, pp. 436–452.

[51] SPARQL, W3c recommendation: SPARQL query language for RDF, 2012. URL http://www.w3.org/TR/rdf-sparql-query/.

[52] J. Widom, S. Ceri, Active Database Systems: Tiggers and Rules for Advanced Database Processing, Morgan Kaufmann Pub, 1996.

[53] I. Motakis, C. Zaniolo, Composite temporal events in active database rules: a logic-oriented approach, in: Deductive and Object-Oriented Databases, Springer, 1995, pp. 19–37.

[54] R. Snodgrass, I. Ahn, Temporal databases, Computer 19 (9) (1986) 35–42.

[55] G. Lausen, B. Ludäscher, W. May, On active deductive databases: the statelog approach, in: Transactions and Change in Logic Databases, Springer, 1998, pp. 69–106.

[56] D. Anicic, P. Fodor, S. Rudolph, R. Stühmer, N. Stojanovic, R. Studer, A rule-based language for complex event processing and reasoning, in: Web Reasoning and Rule Systems, Springer, 2010, pp. 42–57.

[57] F. Bry, M. Eckert, Rule-based composite event queries: the language xchangeeq and its semantics, in: Web Reasoning and Rule Systems, Springer, 2007, pp. 16–30.

[58] R. Ramakrishnan, J.D. Ullman, A survey of deductive database systems, J. Log. Program. 23 (2) (1995) 125–149.

[59] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995, Available online at http://webdam.inria.fr/Alice/.

[60] D.F. Barbieri, D. Braga, S. Ceri, E. Della Valle, M. Grossniklaus, C-SPARQL: SPARQL for continuous querying, in: Proceedings of the 18th International Conference on World Wide Web, WWW '09, ACM, New York, NY, USA, 2009, pp. 1061–1062.

[61] D.F. Barbieri, D. Braga, S. Ceri, M. Grossniklaus, An execution environment for C-SPARQL queries, in: Proceedings of the 13th International Conference on Extending Database Technology, EDBT '10, ACM, New York, NY, USA, 2010, pp. 441–452.

[62] D. Barbieri, D. Braga, S. Ceri, E. Della Valle, M. Grossniklaus, Incremental reasoning on streams and rich background knowledge, in: L. Aroyo, G. Antoniou, E. Hyvnen, A. ten Teije, H. Stuckenschmidt, L. Cabral, T. Tudorache (Eds.), The Semantic Web: Research and Applications, in: Lecture Notes in Computer Science, vol. 6088, Springer, Berlin, Heidelberg, 2010, pp. 1–15.

[63] D. Dell'Aglio, E. Della Valle, Ch. Incremental reasoning on linked data streams, in: Linked Data Management, in: Emerging Directions in Database Systems and Applications, CRC Press, 2014.

[64] Y. Ren, J. Pan, Y. Zhao, Towards scalable reasoning on ontology streams via syntactic approximation, in: Proc. of IWOD, 2010.

[65] Y. Ren, J.Z. Pan, Optimising ontology stream reasoning with truth maintenance system, in: Proceedings of the 20th ACM International Conference on Information and Knowledge Management, ACM, New York, NY, USA, 2011, pp. 831–836.

[66] D. Barbieri, D. Braga, S. Ceri, E.D. Valle, Y. Huang, V. Tresp, A. Rettinger, H. Wermser, Deductive and inductive stream reasoning for semantic social media analytics, IEEE Intell. Syst. 25 (6) (2010) 32–41.

[67] J. Hoeksema, S. Kotoulas, High-performance distributed stream reasoning using s4, in: Ordring Workshop at ISWC, 2011.

[68] D. Le-Phuoc, M. Dao-Tran, J. Xavier Parreira, M. Hauswirth, A native and adaptive approach for unified processing of linked streams and linked data, in: L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. Noy, E. Blomqvist (Eds.), The Semantic Web—ISWC 2011, in: Lecture Notes in Computer Science, vol. 7031, Springer, Berlin, Heidelberg, 2011, pp. 370–388.

[69] A. Bolles, M. Grawunder, J. Jacobi, Streaming SPARQL—extending SPARQL to process data streams, in: S. Bechhofer, M. Hauswirth, J. Hoffmann, M. Koubarakis (Eds.), The Semantic Web: Research and Applications, in: Lecture Notes in Computer Science, vol. 5021, Springer, Berlin, Heidelberg, 2008, pp. 448–462.

[70] O. Walavalkar, A. Joshi, T. Finin, Y. Yesha, Streaming knowledge bases, in: In International Workshop on Scalable Semantic Web Knowledge Base Systems, 2008.

[71] S. Komazec, D. Cerri, D. Fensel, Sparkwave: continuous schema-enhanced pattern matching over RDF data streams, in: Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems, DEBS '12, ACM, New York, NY, USA, 2012, pp. 58–68.

[72] D. Anicic, P. Fodor, S. Rudolph, N. Stojanovic, EP-SPARQL: a unified language for event processing and stream reasoning, in: Proceedings of the 20th International Conference on World Wide Web, ACM, New York, NY, USA, 2011, pp. 635–644.

[73] M. Gebser, T. Grote, R. Kaminski, P. Obermeier, O. Sabuncu, T. Schaub, Stream reasoning with answer set programming: preliminary report, in: Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning, KR 2012, AAAI Press, 2012.

[74] A. Rodrıguez, R. McGrath, Y. Liu, J. Myers, I. Urbana-Champaign, Semantic management of streaming data, Proc. Semant. Sensor Netw. 80 (2009).

[75] C. Gutierrez, C.A. Hurtado, A. Vaisman, Introducing time into RDF, IEEE Trans. Knowl. Data Eng. 19 (2) (2007) 207–218.

[76] T. Scharrenbach, J. Urbani, A. Margara, E. Valle, A. Bernstein, Seven commandments for benchmarking semantic flow processing systems, in: P. Cimiano, O. Corcho, V. Presutti, L. Hollink, S. Rudolph (Eds.), The Semantic Web: Semantics and Big Data, in: Lecture Notes in Computer Science, vol. 7882, Springer, Berlin, Heidelberg, 2013, pp. 305–319. URL: http://dx.doi.org/10.1007/978-3-642-38288-8_21.

[77] D. Le-Phuoc, M. Dao-Tran, M.-D. Pham, P. Boncz, T. Eiter, M. Fink, Linked stream data processing engines: facts and figures, in: The Semantic Web—ISWC 2012, Springer, 2012, pp. 300–312.

[78] D. Dell'Aglio, M. Balduini, E. Della Valle, On the need to include functional testing in RDF stream engine benchmarks, in: 1st International Workhop On Benchmarking RDF Systems, BeRSys 2013, 2013.

[79] D. DellAglio, J.-P. Calbimonte, M. Balduini, O. Corcho, E. Della Valle, On correctness in RDF stream processor benchmarking, in: H. Alani, L. Kagal, A. Fokoue, P. Groth, C. Biemann, J. Parreira, L. Aroyo, N. Noy, C. Welty, K. Janowicz (Eds.), The Semantic Web ISWC 2013, in: Lecture Notes in Computer Science, vol. 8219, Springer, Berlin, Heidelberg, 2013, pp. 326–342. URL: http://dx.doi.org/10.1007/978-3-642-41338-4_21.

[80] D.F. Barbieri, D. Braga, S. Ceri, E. Della Valle, M. Grossniklaus, C-SPARQL: a continuous query language for RDF data streams, Int. J. Semant. Comput. 04 (1) (2010) 3–25.

[81] J. Broekstra, A. Kampman, F. van Harmelen, Sesame: a generic architecture for storing and querying RDF and RDF schema, in: I. Horrocks, J. Hendler (Eds.), The Semantic Web—ISWC 2002, in: Lecture Notes in Computer Science, vol. 2342, Springer, Berlin, Heidelberg, 2002, pp. 54–68.

[82] E. Della Valle, I. Celino, D. Dell'Aglio, K. Kim, Z. Huang, V. Tresp, W. Hauptmann, Y. Huang, R. Grothmann, Urban Computing: a challenging problem for semantic technologies, in: Workshop on New forms of Reasoning for the Semantic Web: Scalable, Tolerant and Dynamic, NeFoRS 2008, vol. 12 2008.

[83] R. Volz, S. Staab, B. Motik, Incrementally maintaining materializations of ontologies stored in logic databases, in: S. Spaccapietra, E. Bertino, S. Jajodia, R. King, D.i. McLeod, M. Orlowska, L. Strous (Eds.), Journal on Data Semantics II, in: Lecture Notes in Computer Science, vol. 3360, Springer, Berlin, Heidelberg, 2005, pp. 1–34.
[84] B. Cuenca Grau, C. Halaschek-Wiener, Y. Kazakov, History matters: incremental ontology reasoning using modules, in: K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, P. Cudr-Mauroux (Eds.), The Semantic Web, in: Lecture Notes in Computer Science, vol. 4825, Springer, Berlin, Heidelberg, 2007, pp. 183–196. URL: http://dx.doi.org/10.1007/978-3-540-76298-0_14.
[85] Lehigh University Benchmark (LUBM), Lehigh University benchmark (LUBM), 2012. URL: http://swat.cse.lehigh.edu/projects/lubm/.
[86] D. Barbieri, D. Braga, S. Ceri, E. Della Valle, M. Grossniklaus, Continuous queries and real-time analysis of social semantic data with C-SPARQL, in: Proceedings of Social Data on the Web Workshop at the 8th International Semantic Web Conference, vol. 10, 2009.
[87] C. Bizer, A. Schultz, The Berlin SPARQL benchmark, Int. J. Semant. Web Inf. Syst. (IJSWIS) 5 (2) (2009) 1–24.
[88] C. Bizer, T. Heath, T. Berners-Lee, Linked data-the story so far, Int. J. Semant. Web Inf. Syst. (IJSWIS) 5 (3) (2009) 1–22.
[89] J. Sequeda, O. Corcho, Linked Stream Data: A Position Paper, 2009.
[90] M. Schmidt, T. Hornung, G. Lausen, C. Pinkel, Sp2bench: a SPARQL performance benchmark, in: IEEE 25th International Conference on Data Engineering, ICDE'09, IEEE, 2009, pp. 222–233.
[91] D. Le-Phuoc, H. Nguyen Mau Quoc, C. Le Van, M. Hauswirth, Elastic and scalable processing of linked stream data in the cloud, in: H. Alani, L. Kagal, A. Fokoue, P. Groth, C. Biemann, J. Parreira, L. Aroyo, N. Noy, C. Welty, K. Janowicz (Eds.), The Semantic Web ISWC 2013, in: Lecture Notes in Computer Science, vol. 8218, Springer, Berlin, Heidelberg, 2013, pp. 280–297. URL: http://dx.doi.org/10.1007/978-3-642-41335-3_18.
[92] S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S.R. Madden, F. Reiss, M.A. Shah, Telegraphcq: continuous dataflow processing, in: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, SIGMOD '03, ACM, New York, NY, USA, 2003, p. 668.
[93] C.L. Forgy, Rete: a fast algorithm for the many pattern/many object pattern match problem, Artificial Intelligence 19 (1) (1982) 17–37.
[94] S. Chakravarthy, V. Krishnaprasad, E. Anwar, S.-K. Kim, Composite events for active databases: semantics, contexts and detection, in: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994, pp. 606–617.
[95] D. Anicic, S. Rudolph, P. Fodor, N. Stojanovic, Real-time complex event recognition and reasoning—a logic programming approach, Appl. Artif. Intell. 26 (1–2) (2012) 6–57.
[96] D. Anicic, P. Fodor, S. Rudolph, R. Stühmer, N. Stojanovic, R. Studer, Etalis: rule-based reasoning in event processing, in: S. Helmer, A. Poulovassilis, F. Xhafa (Eds.), Reasoning in Event-Based Distributed Systems, in: Studies in Computational Intelligence, vol. 347, Springer, Berlin, Heidelberg, 2011, pp. 99–124.
[97] Esper, 2013. http://esper.codehaus.org.
[98] Alert, Alert project, 2012. URL http://www.alert-project.eu/.
[99] A. Mileo, A. Abdelrahman, S. Policarpio, M. Hauswirth, Streamrule: a nonmonotonic stream reasoning system for the semantic web, in: Web Reasoning and Rule Systems, Springer, 2013, pp. 247–252.
[100] NEXRAD Radar, Nexrad radar, 2013. URL: http://en.wikipedia.org/wiki/NEXRAD.
[101] B. Motik, Representing and querying validity time in RDF and OWL: a logic-based approach, J. Web Sem. 12 (2012) 3–21.
[102] M. Perry, P. Jain, A.P. Sheth, Sparql-st: extending SPARQL to support spatiotemporal queries, in: N. Ashish, A.P. Sheth (Eds.), Geospatial Semantics and the Semantic Web, in: Semantic Web and Beyond, vol. 12, Springer, US, 2011, pp. 61–86.
[103] F. Grandi, T-SPARQL: a TSQL2-like temporal query language for RDF, in: International Workshop on on Querying Graph Structured Data, 2010, pp. 21–30.
[104] M. Koubarakis, K. Kyzirakos, Modeling and querying metadata in the semantic sensor web: the model STRDF and the query language STSPARQL, in: L. Aroyo, G. Antoniou, E. Hyvnen, A. ten Teije, H. Stuckenschmidt, L. Cabral, T. Tudorache (Eds.), The Semantic Web: Research and Applications, in: Lecture Notes in Computer Science, vol. 6088, Springer, Berlin, Heidelberg, 2010, pp. 425–439.
[105] J. Tappolet, A. Bernstein, Applied temporal RDF: efficient temporal querying of RDF data with SPARQL, in: L. Aroyo, P. Traverso, F. Ciravegna, P. Cimiano, T. Heath, E. Hyvnen, R. Mizoguchi, E. Oren, M. Sabou, E. Simperl (Eds.), The Semantic Web: Research and Applications, in: Lecture Notes in Computer Science, vol. 5554, Springer, Berlin, Heidelberg, 2009, pp. 308–322.
[106] C. Hurtado, A. Vaisman, Reasoning with temporal constraints in RDF, in: J. Alferes, J. Bailey, W. May, U. Schwertel (Eds.), Principles and Practice of Semantic Web Reasoning, in: Lecture Notes in Computer Science, vol. 4187, Springer, Berlin, Heidelberg, 2006, pp. 164–178.
[107] J. Hoffart, F.M. Suchanek, K. Berberich, G. Weikum, Yago2: a spatially and temporally enhanced knowledge base from wikipedia, Artif. Intell. (2012).
[108] H. Stuckenschmidt, S. Ceri, E.D. Valle, F. van Harmelen, Towards expressive stream reasoning, in: K. Aberer, A. Gal, M. Hauswirth, K.-U. Sattler, A.P. Sheth (Eds.), Semantic Challenges in Sensor Networks, in: Dagstuhl Seminar Proceedings, vol. 10042, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, Dagstuhl, Germany, 2010.
[109] G. Unel, D. Roman, Stream reasoning: a survey and further research directions, in: T. Andreasen, R. Yager, H. Bulskov, H. Christiansen, H. Larsen (Eds.), Flexible Query Answering Systems, in: Lecture Notes in Computer Science, vol. 5822, Springer, Berlin, Heidelberg, 2009, pp. 653–662.
[110] Y. Zhang, P.M. Duc, O. Corcho, J.-P. Calbimonte, Srbench: a streaming RDF/SPARQL benchmark, in: The Semantic Web—ISWC 2012, Springer, 2012, pp. 641–657.
[111] E. Della Valle, S. Schlobach, M. Krötzsch, A. Bozzon, S. Ceri, I. Horrocks, Order matters! Harnessing a world of orderings for reasoning over massive data, Semant. Web J. 4 (2) (2012) 219–231.
[112] J. Urbani, S. Kotoulas, E. Oren, F. van Harmelen, Scalable distributed reasoning using mapreduce, in: A. Bernstein, D. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, K. Thirunarayan (Eds.), The Semantic Web—ISWC 2009, in: Lecture Notes in Computer Science, vol. 5823, Springer, Berlin, Heidelberg, 2009, pp. 634–649.
[113] J. Urbani, S. Kotoulas, J. Maassen, F. van Harmelen, H. Bal, OWL reasoning with webpie: calculating the closure of 100 billion triples, in: L. Aroyo, G. Antoniou, E. Hyvnen, A. ten Teije, H. Stuckenschmidt, L. Cabral, T. Tudorache (Eds.), The Semantic Web: Research and Applications, in: Lecture Notes in Computer Science, vol. 6088, Springer, Berlin, Heidelberg, 2010, pp. 213–227.
[114] M. Ali, An introduction to microsoft SQL server streaminsight, in: Proceedings of the 1st International Conference and Exhibition on Computing for Geospatial Research & Application, ACM, 2010, p. 66.
[115] J.F. Allen, Maintaining knowledge about temporal intervals, Commun. ACM 26 (11) (1983) 832–843.
[116] W. White, M. Riedewald, J. Gehrke, A. Demers, What is "next" in event processing? in: PODS, ACM, New York, NY, USA, 2007, pp. 263–272.
[117] U. Srivastava, J. Widom, Flexible time management in data stream systems, in: Proceedings of the Twenty-Third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, ACM, 2004, pp. 263–274.
[118] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Commun. ACM 21 (7) (1978) 558–565.
[119] W3C RSP, W3c RDF stream processing (RSP) community group, 2013. URL: http://www.w3.org/community/rsp/.
[120] N. Dindar, B. Güç, P. Lau, A. Ozal, M. Soner, N. Tatbul, Dejavu: declarative pattern matching over live and archived streams of events, in: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, SIGMOD '09, ACM, New York, NY, USA, 2009, pp. 1023–1026.
[121] F. Reiss, K. Stockinger, K. Wu, A. Shoshani, J.M. Hellerstein, Enabling real-time querying of live and historical stream data, in: Proceedings of the 19th International Conference on Scientific and Statistical Database Management, IEEE Computer Society, Washington, DC, USA, 2007, p. 28.
[122] A. Deligiannakis, Y. Kotidis, N. Roussopoulos, Compressing historical information in sensor networks, in: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, SIGMOD '04, ACM, New York, NY, USA, 2004, pp. 527–538.
[123] G. Cugola, A. Margara, M. Matteucci, G. Tamburrelli, Introducing uncertainty in complex event processing: model, implementation, and validation, Comput. J. (2012) submitted for publication.
[124] T.J. Green, V. Tannen, Models for incomplete and probabilistic information, IEEE Data Eng. Bull. 29 (2006).
[125] S. Wasserkrug, A. Gal, O. Etzion, Y. Turchin, Efficient processing of uncertain events in rule-based systems, IEEE Trans. Knowl. Data Eng. 24 (1) (2012) 45–58.
[126] C. Ré, J. Letchner, M. Balazinksa, D. Suciu, Event queries on correlated probabilistic streams, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD '08, ACM, New York, NY, USA, 2008, pp. 715–728.
[127] S. Wasserkrug, A. Gal, O. Etzion, Y. Turchin, Complex event processing over uncertain data, in: Proceedings of the Second International Conference on Distributed Event-Based Systems, DEBS '08, ACM, New York, NY, USA, 2008, pp. 253–264.
[128] Y. Diao, B. Li, A. Liu, L. Peng, C. Sutton, T.T.L. Tran, M. Zink, Capturing data uncertainty in high-volume stream processing, in: CIDR 2009, Fourth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4–7, 2009, Online Proceedings.
[129] A. Nikolov, V. Uren, E. Motta, A. Roeck, Integration of semantically annotated data by the knofuss architecture, in: A. Gangemi, J. Euzenat (Eds.), Knowledge Engineering: Practice and Patterns, in: Lecture Notes in Computer Science, vol. 5268, Springer, Berlin, Heidelberg, 2008, pp. 265–274. URL: http://dx.doi.org/10.1007/978-3-540-87696-0_24.
[130] M. Mendler, S. Scheele, Towards constructive DL for abstraction and refinement, J. Automat. Reason. 44 (2010) 207–243.
[131] P. Doherty, J. Gustafsson, L. Karlsson, J. Kvarnström, Tal: temporal action logics language specification and tutorial, Comput. Inf. Sci. 3 (015) (1998).
[132] J.J. Elgot-Drapkin, Step-logic: reasoning situated in time, Ph.D. thesis, College Park, MD, USA, aAI8912283, 1988.
[133] J. Elgot-Drapkin, S. Kraus, M. Miller, M. Nirkhe, D. Perlis, Active Logics: A Unified Formal Approach to Episodic Reasoning, 1999.
[134] R. Haenni, Unifying logical and probabilistic reasoning, in: Symbolic and Quantitative Approaches to Reasoning with Uncertainty, Springer, 2005, pp. 788–799.
[135] Y.E. Ioannidis, Query optimization, ACM Comput. Surv. 28 (1) (1996) 121–123.
[136] J. Callan, Distributed information retrieval, in: W. Croft (Ed.), Advances in Information Retrieval, in: The Information Retrieval Series, vol. 7, Springer, US, 2002, pp. 127–150.
[137] R. Buyya, M. Pathan, A. Vakali, Content Delivery Networks, Vol. 9, Springer, 2008.

[138] A. Carzaniga, A. Wolf, Content-based networking: a new communication infrastructure, in: Developing an Infrastructure for Mobile and Wireless Systems, 2002, pp. 59–68.
[139] A. Hogan, A. Harth, A. Polleres, Saor: authoritative reasoning for the web, in: Proceedings of the 3rd Asian Semantic Web Conference on the Semantic Web, ASWC '08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 76–90.
[140] Y. Ren, J.Z. Pan, K. Lee, Parallel ABox reasoning of EL ontologies, in: Proc. of the First Joint International Conference of Semantic Technology, JIST 2011, 2011.
[141] M. Salvadores, G. Correndo, S. Harris, N. Gibbins, N. Shadbolt, The design and implementation of minimal RDFS backward reasoning in 4store, in: G. Antoniou, M. Grobelnik, E. Simperl, B. Parsia, D. Plexousakis, P. De Leenheer, J. Pan (Eds.), The Semanic Web: Research and Applications, in: Lecture Notes in Computer Science, vol. 6644, Springer, Berlin, Heidelberg, 2011, pp. 139–153.
[142] S. Kotoulas, E. Oren, F. van Harmelen, Mind the data skew: distributed inferencing by speeddating in elastic regions, in: Proceedings of the 19th International Conference on World Wide Web, WWW '10, ACM, New York, NY, USA, 2010, pp. 531–540.
[143] R. Soma, V.K. Prasanna, Parallel inferencing for OWL knowledge bases, in: Proceedings of the 2008 37th International Conference on Parallel Processing, ICPP '08, IEEE Computer Society, Washington, DC, USA, 2008, pp. 75–82.
[144] P. Alvaro, T. Condie, N. Conway, K. Elmeleegy, J.M. Hellerstein, R.C. Sears, Boom: data-centric programming in the datacenter, EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-113, 2009.
[145] A. Gupta, I.S. Mumick, Maintenance of materialized views: problems, techniques, and applications, Data Eng. Bull. 18 (2) (1995) 3–18.
[146] A. Gupta, I.S. Mumick, V.S. Subrahmanian, Maintaining views incrementally, in: ACM SIGMOD Record, vol. 22, ACM, 1993, pp. 157–166.
[147] J.V. Harrison, S. Dietrich, Maintenance of materialized views in a deductive database: an update propagation approach, in: Workshop on Deductive Databases, JICSLP, 1992, pp. 56–65.
[148] J. Urbani, A. Margara, C. Jacobs, F. van Harmelen, H. Bal, DynamiTE: parallel materialization of dynamic RDF data, in: The Semantic Web—ISWC 2013, 2013.
[149] P. Hitzler, D. Vrandei, Resolution-based approximate reasoning for OWL DL, in: Y. Gil, E. Motta, V. Benjamins, M. Musen (Eds.), The Semantic Web ISWC 2005, in: Lecture Notes in Computer Science, vol. 3729, Springer, Berlin, Heidelberg, 2005, pp. 383–397.
[150] T.K. Sellis, Multiple-query optimization, ACM Trans. Database Syst. 13 (1) (1988) 23–52.
[151] G. Cugola, A. Margara, Low latency complex event processing on parallel hardware, J. Parallel Distrib. Comput. 72 (2) (2012) 205–218.
[152] L. Woods, J. Teubner, G. Alonso, Complex event detection at wire speed with FPGAs, Proc. VLDB Endow. 3 (1–2) (2010) 660–669.
[153] M. Sadoghi, M. Labrecque, H. Singh, W. Shum, H.-A. Jacobsen, Efficient event processing through reconfigurable hardware for algorithmic trading, Proc. VLDB Endow. 3 (1–2) (2010) 1525–1528.
[154] N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack, M. Stonebraker, Load shedding in a data stream manager, in: VLDB '2003: Proceedings of the 29th International Conference on Very Large Data Bases, VLDB Endowment, 2003, pp. 309–320.
[155] B. Babcock, M. Datar, R. Motwani, Load shedding for aggregation queries over data streams, in: ICDE '04: Proceedings of the 20th International Conference on Data Engineering, IEEE Computer Society, Washington, DC, USA, 2004, p. 350.
[156] U. Srivastava, J. Widom, Memory-limited execution of windowed stream joins, in: VLDB '04: Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB Endowment, 2004, pp. 324–335.
[157] S. Chandrasekaran, M. Franklin, Remembrance of streams past: overload-sensitive management of archived streams, in: VLDB '04: Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB Endowment, 2004, pp. 348–359.
[158] N. Tatbul, S. Zdonik, Window-aware load shedding for aggregation queries over data streams, in: VLDB '06: Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB Endowment, 2006, pp. 799–810.
[159] Y. Chi, H. Wang, P.S. Yu, Loadstar: load shedding in data stream mining, in: VLDB '05: Proceedings of the 31st International Conference on Very Large Data Bases, VLDB Endowment, 2005, pp. 1302–1305.
[160] S. Schneider, H. Andrade, B. Gedik, A. Biem, K.-L. Wu, Elastic scaling of data parallel operators in stream processing, in: IEEE International Symposium on Parallel & Distributed Processing, IPDPS 2009, IEEE, 2009, pp. 1–12.
[161] B. Satzger, W. Hummer, P. Leitner, S. Dustdar, Esc: towards an elastic stream computing platform for the cloud, in: 2011 IEEE International Conference on Cloud Computing (CLOUD), IEEE, 2011, pp. 348–355.
[162] A. Ishii, T. Suzumura, Elastic stream computing with clouds, in: 2011 IEEE International Conference on Cloud Computing (CLOUD), IEEE, 2011, pp. 195–202.
[163] D. Schneider, The microsecond market, IEEE Spectr. 49 (6) (2012) 66–81.
[164] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, Y. Xing, S.B. Zdonik, Scalable distributed stream processing, in: CIDR, 2003.
[165] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, M. Seltzer, Network-aware operator placement for stream-processing systems, in: Proceedings of the 22nd International Conference on Data Engineering, ICDE '06, IEEE Computer Society, Washington, DC, USA, 2006, p. 49.
[166] M. Balazinska, H. Balakrishnan, M. Stonebraker, Contract-based load management in federated distributed systems, in: NSDI'04: Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation, USENIX Association, Berkeley, CA, USA, 2004, p. 15.
[167] Y. Ahmad, U. Çetintemel, Network-aware query processing for stream-based applications, in: VLDB '04: Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB Endowment, 2004, pp. 456–467.
[168] V. Kumar, B.F. Cooper, Z. Cai, G. Eisenhauer, K. Schwan, Resource-aware distributed stream management using dynamic overlays, in: ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, IEEE Computer Society, Washington, DC, USA, 2005, pp. 783–792.
[169] Y. Zhou, B.C. Ooi, K.-L. Tan, J. Wu, Efficient dynamic operator placement in a locally distributed continuous query system, in: OTM Conferences (1), 2006, pp. 54–71.
[170] L. Amini, N. Jain, A. Sehgal, J. Silber, O. Verscheure, Adaptive control of extreme-scale stream processing systems, in: ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems, IEEE Computer Society, Washington, DC, USA, 2006, p. 71.
[171] T. Repantis, X. Gu, V. Kalogeraki, Synergy: sharing-aware component composition for distributed stream processing systems, in: Middleware '06: Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware, Springer-Verlag New York, Inc., New York, NY, USA, 2006, pp. 322–341.
[172] J. Wolf, N. Bansal, K. Hildrum, S. Parekh, D. Rajan, R. Wagle, K.-L. Wu, L. Fleischer, Soda: an optimizing scheduler for large-scale stream-based distributed computer systems, in: Middleware '08: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware, Springer-Verlag New York, Inc., New York, NY, USA, 2008, pp. 306–325.
[173] R. Khandekar, K. Hildrum, S. Parekh, D. Rajan, J. Wolf, K.-L. Wu, H. Andrade, B. Gedik, Cola: optimizing stream processing applications via graph partitioning, in: Middleware '09: Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware, Springer-Verlag New York, Inc., New York, NY, USA, 2009, pp. 1–20.
[174] G. Cugola, A. Margara, Raced: an adaptive middleware for complex event detection, in: ARM '09: Proceedings of the 8th International Workshop on Adaptive and Reflective Middleware, ACM, New York, NY, USA, 2009, pp. 1–6.
[175] G. Cugola, A. Margara, Deployment strategies for distributed complex event processing, Computing (2012) 1–28.
[176] M. Akdere, U. Çetintemel, N. Tatbul, Plan-based complex event detection across distributed sources, Proc. VLDB Endow. 1 (1) (2008) 66–77.