Seven Commandments for Benchmarking Semantic Flow Processing Systems[⋆]

Thomas Scharrenbach[1], Jacopo Urbani[2], Alessandro Margara[2], Emanuele Della Valle[3], Abraham Bernstein[1]

[1] University of Zurich, Switzerland
scharrenbach@ifi.uzh.ch
[2] Vrije Universiteit Amsterdam, The Netherlands
jacopo@cs.vu.nl, a.margara@vu.nl
[3] Politecnico di Milano, Italy
t emanuele.dellavalle@polimi.it

Abstract. Over the last few years, the processing of dynamic data has gained increasing attention in the Semantic Web community. This led to the development of several stream reasoning systems that enable on-the-fly processing of semantically annotated data that changes over time. Due to their streaming nature, analyzing such systems is extremely difficult. Currently, their evaluation is conducted under heterogeneous scenarios, hampering their comparison and an understanding of their benefits and limitations. In this paper, we strive for a better understanding of the key challenges that these systems must face and define a generic methodology to evaluate their performance. Specifically, we identify three *Key Performance Indicators* and seven *commandments* that specify how to design the stress tests for system evaluation.

1 Introduction

The processing of dynamic data is becoming an important research area in the Semantic Web community, and this is fueled by an increasing number of use cases where input data cannot be considered as static, but rather as a "flow" that continuously changes as computation takes place [16]. Examples range from information produced by on-line newspapers, blogs, and social networks to data generated by sensor networks for environmental monitoring, weather forecast, or traffic analysis in big cities, as well as stock prices for financial analysis.

This led to the definition of a number of stream reasoning systems [10, 11, 20] that combine the on-the-fly processing capabilities of Information Flow Processing (IFP) systems [15] with the use of semantically annotated data, in the form

⋆ We would like to express our thanks to Srdjan Komazec for his valuable comments and discussions. The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2011 under grant agreement no 296126, from the Dutch national program COMMIT, and from the Dept. of the Navy under Grant NICOP N62909-11-1-7065 issued by Office of Naval Research Global.

of RDF triples. To avoid bias in terminology and in continuity with the definition of IFP systems, we collectively denote such systems *Semantic Flow Processing* (SFP) systems. Since in SFP scenarios data changes over time, query answers need to be updated to reflect such changes. This fact turns the entire query process upside-down: whilst "traditional" query engines operate on fixed data and changing queries, the SFP scenario evaluates fixed queries on changing data.

Empirical evaluation of systems is a significant challenge in computer science [23, 24]. Due to their complexity and heterogeneity this is especially true for SFP systems. Despite the number of SFP systems presented in literature, their evaluation is still conducted in incomparable and limited scenarios, without addressing a proper definition of the key performance indicators. This complicates (or even prevents) any meta-analysis comparing the different systems to understand their distinctive aspects, benefits, and limitations.

In this paper, we study the problem of *benchmarking SFP systems* with the purpose of better understanding the key challenges that these system must face and defining a generic methodology to evaluate their performance. We base our study upon a recent survey of IFP systems [15], the commandments for benchmarking databases [18], and our analysis of available benchmarks for testing SFP systems [21, 26]. Our study first *identifies the challenges that SFP systems must face*. Starting from these challenges, we discern the *key performance indicators* (KPIs) of SFP systems and introduce *seven commandments* on how to evaluate the performance of SFP systems according to these KPIs.

This work makes no effort towards defining yet another benchmark for evaluating the performance of SFP systems. On the contrary, *we identify as the main contribution a systematic guideline for assessing the KPIs of SFP systems.* Not only this is useful for a systematic evaluation of a concrete benchmarking framework at hand. By identifying the main KPIs for the abstract SFP scenario, our work can be also used for understanding the requirements of concrete applications as well as guide the design and configuration of an SFP system capable of satisfying them.

The paper is structured as follows: Section 2 provides background information on IFP and SFP systems, as well as on frameworks and methodologies for evaluating their performance. Section 3 investigates the main properties of SFP systems, which we use in Section 4 to present the main challenges in the domain. In Section 5 we discuss the most appropriate KPIs and stress tests for the evaluation. Finally, Section 6 summarizes our findings and concludes the paper.

2 Related Work

This section presents related work in the area of IFP and SFP systems, and in the area of benchmarks for flow-processing systems.

Flow Processing Systems. The last years have seen the development of a large number of IFP systems. These process continuous flows of information based on a set of pre-deployed rules or queries to produce and deliver timely responses to interested parties. Despite their common goals, existing systems greatly differ in the language they use to define queries and on the adopted processing mechanisms [15]. Based on these aspects, we can roughly classify them into two main classes: Data Stream Managements Systems (DSMSs) [8] and Complex Event Processing (CEP) systems [22]. Note that there exist hybrid systems that combine features of DSMS and CEP.

DSMSs have been developed by the database community and exploit a processing model that is similar to that of traditional DBMSs. More in particular, they adopt *window* operators to isolate the portions of streams that are relevant for processing and logically operate on these portions using relational algebra operators. This processing model is described in [6] and, despite some differences, it represents the common ground of all DSMSs [1, 2, 13].

CEP systems [3, 12, 14] take a different approach. While DSMSs use relational operators to *transform* input streams, CEP rules *define* higher level information (in the form of composite events) from patterns of primitive events observed from the external environment.

SFP systems extend the IFP domain by considering semantically annotated data, based on the RDF data model. They extend IFP systems by inference mechanisms that reach from simple RDFS inference to supporting the OWL2 profiles.[4] Most SFP systems [10, 11, 20] use the query model of DSMSs, enriching it with the possibility to perform reasoning over streaming data. Only few approaches [5] take a different direction and combine RDF data with the processing model of CEP systems.

Stream Benchmarking. In the following, we first present the Linear Road Benchmark and the Fast Flower Delivery use case—the accepted means to compare DSMSs and CEP systems—and then SR-Bench and the SLD-Bench – the two existing proposals for benchmarking SFP systems.

Linear Road (LR) This benchmark [7] was proposed by groups at MIT, Brandeis University, Brown University, and Stanford University to compare the performance characteristics of different DSMSs and of alternative (e.g., Relational Database) systems. LR simulates a variable tolling system for highways. Toll charges are determined dynamically considering traffic congestion and accident proximity. The benchmark does not specify a solution but describes the requirements of the tolling system both functionally (e.g., how to determine the level of traffic congestion or to detect accidents) and non-functionally (e.g., the vehicle must receive toll notifications at most five seconds after moving from one road segment to the following one). LR comes with a simulator, an environment that validates the results of the system being benchmarked, and a set of software sensors to measure response time and supported query load.

[4] `http://www.w3.org/TR/owl2-profiles/`

Fast Flower Delivery (FFD) evolved from a running example [17] to a must-to-implement showcase for commercial CEPs. It proposes a logistic scenario, where independent van drivers are asked to deliver flowers from the city's flower stores to their destinations. The use case is divided into five phases: *1*) a bid phase, when a store offers highly rated drivers nearby to deliver flowers to a destination within a given time; *2*) an assignment phase, when the system (manually or automatically) decides which driver shall deliver the flowers; *3*) a deliver process, when the system monitors the delivery process; *4*) a ranking evaluation, when the system increases or decreases each driver's ranking based on the ability to deliver flowers on time; and *5*) an activity monitoring, when the system evaluates drivers ranking over time.

SR-Bench (SR) is defined on measurements of sensors and a fixed (i.e. non-parameterized) set of queries [26]; some of which require RDFS reasoning capabilities. Each graph points to a) the sensor, b) the timestamp of the observation, and c) the actual observation. Each of the above refers to a complex object, where the sensor, the timestamp and the observation follow a pre-defined fixed schema. Observations are considered as flow-data whereas the schema and the background knowledge are considered fixed. SR-Bench comprises 17 queries that can be divided in sub-categories to test different kinds of use-cases: 1) query only flow-data (Q1-Q7), 2) query both flow and background data (Q8-Q11), and 3) additionally query the GeoNames and DBpedia datasets (Q12-Q17). Some of these queries require inference capabilities (Q3, Q15-17).

SLD-Bench [21] is defined on three synthetically generated social streams (i.e., a stream of GPS position of the social media users, a stream of microposts, and a stream of uploaded images), a synthetically generated social graph, and a fixed (i.e., non-parameterized) set of queries. Emphasis is on processing social streams against a large dataset of static data. SLD-Bench includes 12 queries: some challenge only flow data (Q1, Q4, Q8, Q10-Q11), others joining flow and static data (Q2, Q3, Q5-Q7, Q9), none requiring inference capabilities.

3 Properties of SFP Systems

Following the terminology for IFP systems [15] Figure 1 shows the abstract architecture of an SFP system. It receives flows (or streams) of *information items* from external sources and processes them on-the-fly to produce results for a set of connected *sinks*. All existing SFP systems use RDF triples for representing information items. Processing is governed by a set of *rules* or *queries* deployed into the system. It is performed by one or more interconnected *processors* and may consider (semi)static *background data* in addition to the information flowing from sources. Processors cooperate to generate final results for sinks by producing and sharing *partial results* (e.g. variable bindings that are not yet complete).

With reference to this architecture, we identified *seven* main properties of SFP system. Note that these properties are not unique, but rather those useful to determine the list of challenges for an SFP system. A complete classification of SFP systems is however beyond the scope of this paper.

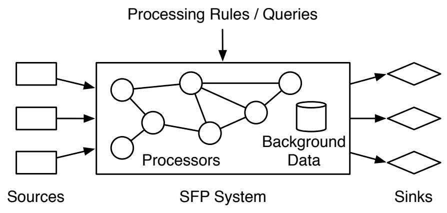

Fig. 1. Abstract Architecture of an SFP System

[P1] Support of Background Data. It defines the feature of considering background data during processing. An SFP system can either support or ignore background data; assume that such data is fixed and available ex-ante; or allow (infrequent) changes to this data.

[P2] Inference Support. The usage of semantically annotated data allows the SFP system to infer implicit information. This process is broadly referred as *inference* or *reasoning*. The ability of performing inference is feature unique to SFP system and not available in IFP systems. We make, however, no assumption about the expressive power of the inference mechanism.

[P3] Quality of Service (QoS). The QoS property identifies whether an SFP system performs best effort processing or guarantees some specific levels of performance. The two main metrics to measure QoS for SFP systems are completeness and soundness of results along with the response time. Completeness measures whether the system guarantees a certain proportion of all correct answers, while soundness measures the number of incorrect results due, for example, to approximation.

[P4] Time Model. In flow-processing applications time plays a central role. Information items are situated in time and an SFP system may provide time for each data item either explicitly or implicitly. In the first case, time is explicitly present in the data-flow while in the latter case the system assigns some timestamp or interval to each incoming item. Current SFP systems either encode time using RDF (by using an RDF node), or add a timestamp or interval to information items, which thus become quads or quintuples instead of triples.

[P5] Time Semantics. Time can be modeled using point-based semantics or interval-based semantics. The point based semantics associates each information item in the data-flow a single point in time (e.g. the occurrence of the event or the incoming time in the system). In contrast, interval-based semantics defines an interval of validity for the associated information.

[P6] Query Model. In the context of SFP, the query model is a discriminating property between systems. Systems like EP-SPARQL [5] define *pattern matching* queries through a set of primitive operators (e.g. sequences). Conversely, systems like C-SPARQL [10] extend *declarative languages* like SQL, augmenting them with operators like windows to limit the scope of processing. The query model also defines when queries are evaluated. The evaluation can be either reactive (the query is triggered when new data arrives), or periodic (the query is executed at a specified interval of time).

[P7] Distribution. To better support large scale scenarios, with sources of information potentially distributed over a wide geographical area, SFP systems may enable processors (see Figure 1) to be distributed among different physical machines. Distribution enables the concurrent execution of different queries at different nodes, but also the incremental evaluation of the building blocks of a single query on different machines. In the latter case, distribution can be used to push filtering operators as close as possible to the sources of the streaming information, to reduce the volume of data propagated over the network.

4 Challenges

Defining the challenges we rely on the following assumptions. They clearly define the scope of our analysis, and thus the area of validity of our results.

- **SFP systems distinguish between stream data and background data.** Stream data changes at high frequency while background data is static.
- **Streamed data does not affect the schema; no schema information is present in the stream.** In the Semantic Web, schema is defined by ontologies describing a conceptualization for a domain of interest. Since SFP systems assume that schema information does not change frequently it is not present in the stream. Note that this does not contradict the Semantic Web's Open-World-Assumption: an SFP system's inference process may still discover new schema statements as long as the reasoning remains monotone.
- **Only deductive and analytical processing is considered.** To limit the scope of our paper, we do not consider inductive processing (e.g., inductive reasoning). It is based on completely different methods and therefore introduces new challenges and requires a separate evaluation methodology.

We identified five classes of challenges that affect both the design and the development of a SFP system: *Managing Background Data*, *Inference Expressivity*, *Time Modeling*, and *Querying*, *Managing Bursts*. Each of them relate to one or more properties of the SFP systems, as shown in Figure 2.

[C1] Managing Background Data. Several challenges are connected to handling background data (P1) next to streaming information. First of all, storing and manipulating background data might be difficult due to the *size* of the data, which can stress the machine resources. Data that greatly exceeds the size of main memory, requires algorithms to govern the data transfer between disk and memory.

Additional challenges derive from the complexity of queries over background data (P6). Queries may require to combine, i.e., join large portions of background data together with the elements in the flow. This poses strict timing constraints to processing, thus demanding for the definition and maintenance of suitable data structures for efficient retrieval and processing of information. This may

	C1: Managing Background Data	C2: Inference Expressivity	C3: Time Modeling	C4: Querying	C5: Managing Bursts
P1: Background Data	●	●		●	
P2: Inference Support		●			
P3: Quality of Service		●	●	●	●
P4: Time Model		●	●	●	
P5: Time Semantics		●	●		
P6: Query Model	●			●	
P7: Distribution	●			●	●

Fig. 2. Relations between the challenges and SFP properties

involve changes to the background data, that needs to be timely propagated to processors. Partial results from the flow computation might become invalid, if in the meantime the background data has changed—an even more challenging aspect in the case of parallel and distributed processing (P7).

An SFP system must develop efficient mechanisms to handle all these issues, and their design and implementation is certainly not trivial. Therefore, efficient mechanisms for storing, accessing, and updating background data are crucial and should be properly considered in the evaluation of such systems.

[C2] Expressive Power of Inference. The support for inference (P2) is the distinguishing feature of SFP over IFP systems and introduces serious challenges.

The super-linearity of reasoning (quadratic for RDFS to super-exponential for OWL 2) requires to carefully balance the expressive power of the inference mechanism and performance. Even though inference can be limited to become tractable the fast change rate inherently present in a data-flow imposes strict constraints on the inference process (P3).

Inference requires a frequent interaction between background and stream data (P1), as all SFP systems store schema independent from the flow-data. Efficient mechanisms for storing as well as accessing the schema guarantee fast inference over the flow-data. Entailment regimes like RDFS produce many duplicates requiring an SFP system to handle repeatedly inserted information.

One additional challenge in the inference process is connected with the validity of the information in the system (and this strictly relates it to the properties P4 and P5). If a triple, for example, is no longer valid (e.g., because the active window has moved), then the inference process might have to be repeated to verify whether some conclusions still hold or should be retracted.

[C3] Time Modeling. This challenge differs from the others because it relates to the design of the system while the others primarily affect its execution. In fact, choosing a specific model –and a corresponding semantics– for representing time (P4 and P5) can significantly impact the performance of the system (P3). For example, it has been proven in [25] that the use of an interval-based semantics rather than a point-based semantics may negatively impact the tractability of some time-based operators (e.g., next, sequences). Therefore, the designer of an SFP system must carefully analyze the requisites of the system in order to choose an appropriate time model in order not to jeopardize its performance.

The current RDF data model includes no notion of time, which led existing SFP systems to extend RDF in several ways to handle time, e.g., by timestamping the triples —with potential serious consequences for complex processing tasks such as reasoning on the data. Suppose that the data exploits the RDFS semantics which allows reasoning by an exhaustive application of if-then rules. If the RDF triples used to derive some conclusions are no longer valid, then it is unclear what happens to the derivation. All these uncertainties can be clarified by a formal definition of the model and semantics of time associated to RDF data, but currently there is no clear consensus on this aspect, and this hampers an understanding of the consequences of the processing of SFP systems.

[C4] Querying. The query model determines the processing strategy (P6). A key challenge is its definition for stream and background data (P1) that can satisfy application level requirements on expressive power and ease of use, while keeping the processing as simple and efficient as possible (P3).

An important challenge for CEP-inspired languages is *the choice of an appropriate strategy for storing, accessing, and discarding partial results.* This is even more important when dealing with aggregates, in particular under non-shrinking semantics [9], i.e., when we are not only interested in the number of items in an aggregate but also in the items themselves. (cf. Section 5, S4).

Languages may include *operators that implicitly determine the scope of processing*, e.g., time-constrained sequences. Similarly, in DSMS-inspired transforming languages, the type and size of windows determines the portion of flow-data considered for processing (P4). In both cases, isolating the elements that are relevant for processing is a key challenge. An inappropriate choice may negatively impact the performance of an SFP system: A window too small may never contain enough information to provide the desired results; a window too large may hamper the system's response time. Unsuitable strategies for storing and pruning partial results may further negatively influence response time.

Other challenges rises from the *mechanism for triggering queries* and the *management of multiple queries.* Increasing the frequency of query evaluation may decrease the system's response time while too infrequent evaluations may prevent the detection of critical situations—both resulting in decreasing system performance. SFP systems must be able to develop techniques for sharing the state of partial results that are common to multiple queries, thus reducing memory requirements and processing effort. The effort of managing multiple queries increases in presence of distributed settings (P7) by the necessity of concerting the distribution of operations over available resources with respect to processing capabilities, connectivity, and their geographical location.

[C5] Managing Bursts. SPF systems must be able to continuously provide timely answers to queries even in presence of sudden bursts. This strictly relates to the property P3: indeed, depending from the QoS agreements between the system and the users, it may be acceptable to sacrifice completeness of results for the sake of guaranteeing lower response times. Moreover, managing bursts also requires a careful design of the mapping of processing tasks to available processing components, enabling load balancing and avoiding bottlenecks. This issue becomes even more relevant in parallel and distributed systems (P7).

5 Seven Commandments of Benchmarking SFP Systems

The evaluation of a system performance is done by changing the environment and/or the system parameters and observing the behavior of some measurable Key Performance Indicators (KPIs) as these changes occur. The goal of a benchmark consists in designing a number of stress tests so that the user can measure how different systems react to the same changes, considering the same KPIs. These stress tests should properly create situations when the system is called to deal with the challenges of the domain. The LR benchmark, for example, "is designed to measure how well a system can meet real-time query response requirements in processing high-volume streaming and historical data." [7].

In this section, we first define a number of KPIs to evaluate SFP systems with respect to the challenges identified in Section 4. Then, we design some stress tests to measure and compare the performance of various systems. We thereby analyze to what extent current benchmarking tools cover such stress tests (see Table 1), and provide guidelines how the missing parts can be implemented.

Note that *we provide no unified benchmark but a unified model for systematically benchmarking aspects of SFP systems by stress tests.* An actual implementation of these stress tests will depend first on the actual SFP system and second on the use-cases at hand, and is beyond the scope of this paper.

5.1 Key Performance Indicators

In contrast to offline systems, *SFP systems are reactive.* A delay exists between the points in time when the system consumes an input element and it reports the results of its processing. If the system load exceeds available resources either this delay compromises system reactivity or the system has to drop data.

All benchmarks for SFP systems use throughput as their KPI. This choice yet ignores other criteria that were reported for IFP systems in [15]. We hence identified the following three KPIs as the most suitable regarding our context. Interestingly, they were also used used for the evaluation (yet not benchmarking) of most the principal current SFP systems.

– **Response time** over all queries (Average/x^{th}Percentile/Maximum).
– **Maximum input throughput** in terms of number of data element in the input stream consumed by the system per time unit.
– **Minimum time to accuracy and the minimum time to completion** for all queries [19].[5]

Stressing a system means exploring the input space and identifying best, average, and –most importantly– worst cases for its performance, i.e., the conditions under which the system performs how in relation to the KPIs.

[5] This includes recall, precision and error rate in relation to processing time.

Table 1. Stress tests existing benchmarks support. P indicates a potential support or a partial implementation for stress testing. (a) load balancing, (b) simple, (c) sequential or (d) temporal joins flow-flow data, (e) joins on flow-background data, aggregates under shrinking (f) and non-shrinking semantics (g), (h) out-of-order or (i) missing data, (j) inference, and finally (k) changes in background data.

Benchmark	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**
	(a)	(b) / (c) / (d)	(e)	(f) / (g)	(h) / (i)	(j)	(k)
LR	P	Yes/ Yes/ P	No	No/ Yes	P/ P	No	No
FFD	P	Yes/ P/ Yes	No	No/ Yes	No/ P	P	No
SR	P	Yes/ P/ P	Yes	P/ Yes	P/ P	Yes	P
SLD	P	Yes/ P/ P	Yes	P/ Yes	P/ P	P	P

5.2 Stress Tests

After identifying the KPIs, the definition of stress tests first involves diagnosing *which* parameters to manipulate to change the input of the system. In the case of SFP systems, these parameters have some impact on background data, streaming data, input rate, etc. It is important to devise *how* to change these parameters to achieve the purpose of the test, i.e. to properly impact on the desired KPIs.

In this section we present the *seven commandments* we worked out based on our study of the challenges in Section 4. Each commandment represents one of the stress tests that in our opinion best suit the evaluation of SFP systems. We show how the current benchmarks address these tests in Table 1. We observe that all the benchmarks identified in Section 2 either implemented one or more of these stress tests or could implement them (indicated by "P"). However, no existing benchmark fully implements all of them.

[S1] Load Balancing [Relates to C5]. SFP systems usually consider multiple input flows of information, with possible bursts (C5). Therefore, the SFP system must implement a proper mapping of operators over available processors and good load balancing strategies.

Finding potential bottlenecks in settings in which many queries are deployed and multiple processors are available is extremely difficult. However, benchmarks can empirically evaluate a system under various conditions by repeatedly applying a set of changes to the input. In particular, it is possible to stress the system by (*i*) changing the load of every stream relative to the others at random, (*ii*) creating bursts on an increasing number of input streams, and by (*iii*) dynamically switching data sources to provide their input on some other data flow. All current benchmarks identified in Section 2 provide streaming data from sensors, and therefore implement variants of this stress test. However, the sensors in SR can only emit data on regular stable intervals. SLD and LR offer support for skewed distributions for the generation rate of different streams, although the specifications do not clearly state to what extent the skew can be controlled.

[S2] Joins and Inference on Flow Data Only [Relates to C3, C4]. In order to stress the joins between bindings of flow data we need to distinguish between simple, sequential, and temporal joins. Simple joins put no further constraints on the join but the join-equality. Sequential joins add a sequential constraint (like the `SEQ`-operator [5]). Temporal joins further extend sequential joins by enabling advanced temporal constraints such as Allen's intervals [4]. Note that both sequential and temporal joins require that the system defines an ordering of the flow-data (C3) as well as a proper extension of the query language (C4).

A stress test to measure the performance of data joins has to consider increasingly complex cascades of joins. For testing sequential and temporal joins a benchmark will have to add further constraints on the joins, which have to be reflected in the data. The current benchmarks LR and SLD provide data and use-cases for sequential joins but at the moment none of them implements stress tests for temporal joins—although all datasets would allow to. Therefore, a full implementation of this stress-test is currently unavailable in these benchmarks.

[S3] Joins and Inference in Flow and Background Data [Relates to C1, C4]. In contrast to joins on flow data only, joining stream and background data is not subject to any ordering and hence always results in simple joins. These can be stress-tested by considering single joins and increasingly complex cascades thereof. Notice that systems often exploit the combination of flow and background data to perform inference. In this context, the ability of the system to manage background data (C1) is crucial, since complex reasoning tasks (C4) can require frequent and repeated access to background data.

Currently, both the SR and SLD benchmarks only provide a few fixed queries. They are not parameterized, and thus do not allow an exhaustive assessment of join performance. Furthermore, only SR and SLD can stress an SFP by considering the accesses to background data that is stored in the disk. Conversely, the background data of LR and FFD easily fits into the main memory.

[S4] Aggregates [Relates to C3, C4]. Aggregates enable computation on groups of entities or literals. Such computations include statistics such as counts, averages but also any other arithmetic operation on groups nodes that fulfill a grouping constraint. We distinguish between aggregating over entities and literals. In contrast to literal aggregates, entities aggregates refer to groups of actual entities and not data values. Consider, for example, detecting situations where more than n people with similar interest are watching the same show.

We refer to detecting the sole event as shrinking semantics, i.e., we are *not* interested in the actual people but only some statistics about them. Referring to the actual entities taking part in the aggregate (i.e., the actual people watching the show) is called non-shrinking semantics [9]. We may assess both types by testing *a*) how the system scales with an increasing number of groups (lots of shows, n small), *b*) by increasing the complexity of the grouping constraints (complex definition of similar interests) and *c*) by adjusting the data such that there will be a lot of candidates for groups of which only a small number will finally fulfill the grouping criterion (lots of shows with a number of viewers just below the threshold n).

In contrast to shrinking semantics, non-shrinking semantics are not directly supported by standard SPARQL and also not implemented by any of the existing benchmarks. All of the benchmarks test aggregates in a limited scope, e.g., by implementing single queries (SR, SLD) or the expected outcome (LR).

[S5] Unexpected Data [Relates to C3, C4]. In distributed settings, SFP systems have to deal with out-of-order arrival of information and data loss. This may affect the correctness of query answers, especially (C3) when temporal operators and constraints are involved. We can measure the ability to handle out-of-order observations by (*i*) increasing the number of events arriving not in the expected order; (*ii*) by testing the amount of time or data which can be handled until some out-of-order observation will be no longer considered for processing. SPARQL OPTIONAL operators, for example allow answering a query even if some data is still missing (C4). In both cases the benchmark should measure precision and recall of the amount of missing data. Interestingly, none of the current benchmarks implements tests for out-of-order events or missing data.

While the ability to deal with noisy data is a relevant problem it is our form believe that this must be handled outside the core query processing. Consequently, we did not add stress-tests for handling noisy data.

[S6] Schema [Relates to C1, C2]. Since the schema of both the stream and the background data is known ex-ante, we can only evaluate the system's ability to handle (*i*) an increasing number of statements in the schema (i.e., axioms of the system's ontology), and (*ii*) statements that generate a more complex reasoning. In this last case the system needs to provide inference services (C2).

Number of Axioms. When testing an SFP system by increasing the number of axioms in its ontology it is fundamental to add *new* axioms that could not have been deduced from existing ones. Moreover, the expressive power should not increase as this will spoil the results of this test. SR and SLD are the only benchmarks with ontology schemata. In spite of the several thousand axioms the ontologies comprise the number of axioms involved in these benchmarks' queries is roughly one per cent of that number.

Expressive Power. Increasing the expressive power of the schema not only for the background data but also of the flow data may stress an SFP system significantly [9]. Evaluating the impact of expressive power requires changing the constraints or rules applied by the reasoner, while leaving the ontology unchanged, e.g., by implementing different combinations of the RDFS inference rules or different profiles of OWL 2. The variation in complexity must have some effect on the performance of the inference engine. Adding, for example, disjunction to the reasoner only makes sense in case the ontology contains disjunctive axioms.

In spite of missing features like negation, testing variations of the expressive power is possible in SR and SLD as they refer to some OWL 2-DL ontologies. Currently, they only test *whether* RDFS subclass reasoning is possible but do not measure the impact on KPIs when varying the expressive power. On the other hand, works like [9] provide a stress test for inference on transitive properties under RDFS semantics.

[S7] Changes in Background-Data [Relates to C1, C2] Nearly all systems identified in [15] consider background data in answering queries and this by precompiling the query. When the background data changes (C1), those parts have to be re-compiled and in this intermediate state processing may be delayed or corrupted,[6] further worsened by the presence of inference services (C2). Stress-testing changes in background data should aim at varying the update frequency and the sheer amount of data that is subject to an update. Addressing the query-related part of the background data it should force the system to access background-data from disk as much as possible.

Currently, no benchmark implements this stress test, and only benchmarks that use datasets with rich background data can properly implement it, which is not the case for LR and FFD. The SLD and SR benchmarks do support such data and are suitable for this task. In particular for the SR system, we can simply increase the background data by all those datasets in the LOD cloud for which we may establish links to the GeoNames dataset.

6 Conclusion and Future Work

SFP systems are becoming increasingly popular for processing flows of semantically annotated data on-line and on large scale. Yet, the field of SFP lacks a classification scheme such as [15] for understanding and comparing existing systems. Even more significanlty, there is a lack of common agreement of which are the key performance indicators in the field, and they can be evaluated. A few good proposals for benchmarking SFP systems were published recently [21, 26], but none of them has (yet) come up with a pair of simulator/validator systems comparable to what the LR benchmark provides for IFP systems.

In this paper we diagnosed this research gap and approached the problem of benchmarking SFP systems from another perspective, following a top-down approach. We identified those properties of SFP systems relevant for understanding the key challenges SFP system face and defining the key performance indicators that allow to assess such challenges.

Starting from this analysis, we proposed seven *commandments* for defining a set of benchmarks that comprehensively stress test SFP systems in relation to precisely defined KPIs. We worked out these commandments as currently the most important for benchmarking current SFP systems. With new features for SFP systems this list will certainly have to be extended. For the same reasons as the LR benchmark, we provided no algorithm for implementing a benchmark nor did we address the definition of a common protocol for running a concrete benchmark on different systems. Instead we provide clear guidelines that specify how concrete benchmarks can implement relevant stress tests for SFP systems.

It is our firm belief that following these guidelines will enable implementing new or adjusting existing benchmarks, thus making it possible to realize a thorough evaluation and comparison of SFP systems, clearly spotting their strenghts and weaknesses. The tale of understanding SFP systems by systematic evaluation and comparison has only just begun.

[6] Note that a change in background data does *not* allow for a change in the schema.

References

[1] A data stream language and system designed for power and extensibility. In: Yu, P.S., Tsotras, V.J., Fox, E.A., Liu, B. (eds.) Proc. CIKM'06. pp. 337–346. ACM (2006)

[2] Abadi, D., Carney, D., Çetintemel, U.U.G., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., Zdonik, S.: Aurora: a new model and architecture for data stream management. VLDB J. 12(2), 120–139 (2003)

[3] Agrawal, J., Diao, Y., Gyllstrom, D., Immerman, N.: Efficient pattern matching over event streams. In: Wang, J.T.L. (ed.) Proc. SIGMOD'08. p. 147. ACM (2008)

[4] Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. of the ACM 26(11), 832–843 (Nov 1983)

[5] Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: EP-SPARQL: a unified language for event processing and stream reasoning. In: Srinivasan, S., Ramamritham, K., Kumar, A., Ravindra, M.P., Bertino, E., Kumar, R. (eds.) Proc. WWW'11. pp. 635–644. ACM (2011)

[6] Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Nishizawa, I., Rosenstein, J., Widom, J.: STREAM : The Stanford Stream Data Manager. IEEE Data Eng. Bull. pp. 19–26 (2003)

[7] Arasu, A., Cherniack, M., Galvez, E., Maier, D., Maskey, A.S., Ryvkina, E., Stonebraker, M., Tibbetts, R.: Linear Road : A Stream Data Management Benchmark. In: VLDB J. (2004)

[8] Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Popa, L., Abiteboul, S., Kolaitis, P.G. (eds.) Proc. PODS '02. pp. 1–16. ACM (2002)

[9] Barbieri, D., Braga, D., Ceri, S.: Incremental reasoning on streams and rich background knowledge. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) Proc. ESWC 2010. LNCS, vol. 6088, pp. 1–15 (2010)

[10] Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: C-SPARQL: A Continuous Query Language for RDF Data Streams. Int. J. of Semantic Computing 4(1), 3–25 (2010)

[11] Barbieri, D., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: C-SPARQL: SPARQL for continuous querying. In: Quemada, J., León, G., Maarek, Y.S., Nejdl, W. (eds.) Proc. WWW'09. pp. 1061–1062. ACM (2009)

[12] Brenna, L., Demers, A., Gehrke, J., Hong, M., Ossher, J., Panda, B., Riedewald, M., Thatte, M., White, W.: Cayuga : A High-Performance Event Processing Engine. In: Chan, C.Y., Ooi, B.C., Zhou, A. (eds.) Proc. SIGMOD'07. pp. 1100–1102. ACM (2007)

[13] Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M.J., Hellerstein, J.M., Hong, W., Krishnamurthy, S., Madden, S., Reiss, F., Shah, M.A.: TelegraphCQ: Continuous Dataflow Processing. In: Halevy, A.Y., Ives, Z.G., Doan, A. (eds.) Proc. SIGMOD'03. p. 668. ACM

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

[15] Cugola, G., Margara, A.: Processing Flows of Information: from Data Stream to Complex Event Processing. ACM Comput. Surv. 44(3), 1–62 (Jun 2012)

[16] Della Valle, E., Ceri, S., Milano, P., Van Harmelen, F.: It's a Streaming World! Reasoning upon Rapidly Changing Information. J. Intell. Syst., IEEE (2009)

[17] Etzion, O., Niblett, P.: Event Processing In Action. Manning Publications Co., Greenwich, CT, USA (2010)

[18] Gray, J.: The Benchmark Handbook for Database and Transaction Systems. Morgan Kaufmann, 2nd edn. (1993)

[19] Hellerstein, J.M., Haas, P.J., Wang, H.J.: Online Aggregation. In: Peckham, J. (ed.) Proc. SIGMOD'97. pp. 171–182. ACM (1997)

[20] Le-phuoc, D., Dao-tran, M., Parreira, J.X., Hauswirth, M.: A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) Proc. ISWC 2011. LNCS, vol. 7031, pp. 370–388. Springer Berlin / Heidelberg (2011)

[21] Le-phuoc, D., Dao-tran, M., Pham, M.d.: Linked Stream Data Processing Engines : Facts and Figures. In: Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J.X., Hendler, J., Schreiber, G., Bernstein, A., Blomqvist, E. (eds.) Proc. ISWC 2012. LNCS, vol. 7650, pp. 1–12 (2012)

[22] Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley (2002)

[23] Tichy, W.F., Lukowicz, P., Prechelt, L., Heinz, E.A.: A Quantitative Evaluation Study in Computer Science :. J. Syst. and Softw. 28(1), 9–18 (1995)

[24] Wainer, J., Novoa Barsottini, C.G., Lacerda, D., Magalhães de Marco, L.R.: Empirical evaluation in Computer Science research published by ACM. J. Inform. and Softw. Tech. 51(6), 1081–1085 (Jun 2009)

[25] White, W., Riedewald, M., Gehrke, J., Demers, A.: What is "next" in event processing? In: Libkin, L. (ed.) Proc. PODS '07. pp. 263 – 272. ACM (2007)

[26] Zhang, Y., Duc, P.M., Corcho, O., Calbimonte, J.p.: SRBench : A Streaming RDF / SPARQL Benchmark. In: Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J.X., Hendler, J., Schreiber, G., Bernstein, A., Blomqvist, E. (eds.) Proc. ISWC 2012. LNCS, vol. 1380 (2012)