

Chapter 1

Heaven

In this chapter we present *Heaven*, an open source framework for Systematic Comparative Research on RSP Engine. It consists in four baselines and two main components: the Test Stand and the Analyser. The first section, Section 1.1 introduces the Test Stand, which satisfies requirements from R.1 to R.8 and from R.10 to R.12, by executing experiments on an RSP Engine. Section 1.2 describes the Baselines: four RSP Engines that are included in *Heaven* They fulfil requirements from R.13 to R.14, so are exploitable as naive terms of comparison. In Chapter ?? we use those Baselines to show *Heaven* potential). Finally Section 1.3 presents the Analyser, which addresses requirements R.9 and R.10 allowing the user to visualise, investigate and compare experiment results.

1.1 Test Stand

Aerospace engineering defines an Engine Test Stand as a facility used to develop, study and characterise engines. It allows to test operating regimes and offers measurement of several variables associated with engine workflow. A Test Stand may uses actuators for attaining a specific engine state, which is a unique combination of the engine properties. The information collected through the sensors depends on the engine manufacturer, which usually provides his own stand or the facilities to test the engine with commercial solutions. The Test Stand executes black box analysis, because usually the engines does not allow easily to interact with its internal mechanisms.

A Test Stand can be described similarly in the SR context. The definition above still holds its relevance with the difference that the tested engines are IO-Systems. An RSP Engine consumes an RDF Stream and produces a new one, by applying queries under some entailment regime and w.r.t. some static data (detailed in Section ??). Describe an RSP Engine means understanding the relation between input stream, the queries register to it and what we call "operational semantics", which is strictly implementation dependent and requires to know the RSP Engine internal process. Indeed, a Test Stand provides only black box testing, even having access to the entire RSP Engine code. Anyhow *Heaven* Test Stand allows the user (the RSP Engine developer) to add its own sensor to the execution, respecting requirement R.7 and R.10. In this way is possible to develop a specific testing procedure, which consider also the know-how about this particular engine.

1.1.1 Architecture & Workflow

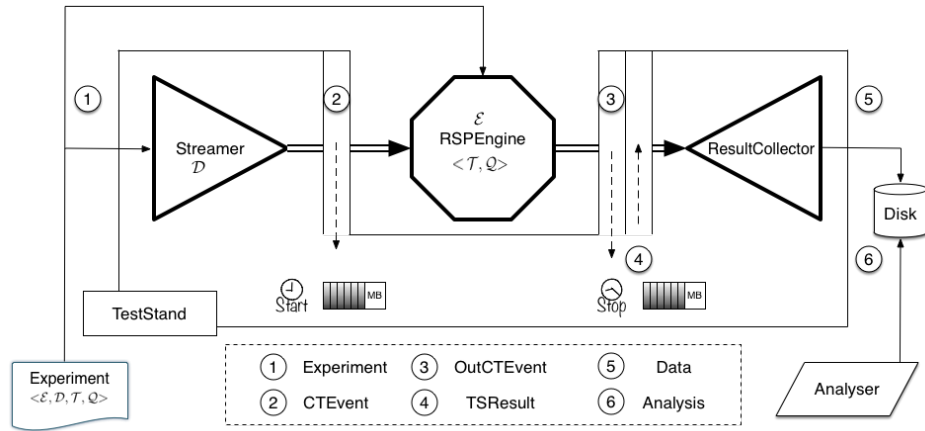


Figure 1.1: *Heaven* modules and workflow

A test stand exploits different modules to simulate the operating regime for the engine in use (i.e a module for fuel distribution, one for the engine mechanic support or to enable users interaction during the execution). *Heaven* TEST STAND is modular, it consists in stand-alone components that can be replaced with ones with the same interfaces [R.10]. At this moment, three modules compose the Test Stand:

- the STREAMER, a source for the input RDF Stream

- the RSP Engine we want to test;
- the RESULT COLLECTOR, a Data acquisition system for both the query results and the gathered measurements.

Figure 1.1 shows both the architecture of *Heaven* TS, how the modules are configured, and its workflow, how the modules interact.

We start presenting the TS architecture, where the components above are arranged into a pipeline and communicates exchanging events [R.11]. The communication starts from the STREAMER, which hides the data generation logic in order to obtain data independence [R.1]. Moreover, it pushes an RDF Stream directly to any mounted RSP Engine, respecting [R.5] by not influence the memory footprint with heavy data loading tasks.

An interface adapts the event flow to the RSP Engine in use, fulfilling [R.2] (Engine Independence) and hiding the query registration process [R.3] (Query independence), which happens at engine level and is up to the RSP Engine provider.

The RESULT COLLECTOR is the tail of the pipeline. It is part of the TEST STAND because the performance measurements are processed and gathered during the execution, together with the queries results data. The evaluation usually happens a-posteriori through the Analyser (Section 1.3). However, real time analysis of the Key Performance Indicator (KPI) are possible, but they may violate some requirements like [R.4 and 5].

Last but not least, the Test Stand has an external structure that sustains other modules. It allows the user to control the process through accessible APIs. This structure can be considered as a module itself, it adds sensor data to query results controlling the execution flow as required by [R.4]. The TEST STAND orchestrates the communication between the upstanding models, forcing the STREAMER to push events to the RSP Engine and the RESULT COLLECTOR to listen the output and collect the results.

To explain the TEST STAND workflow we split the process at the points when the modules exchange events, since each event represent a different logic step in the experiment execution cycle. In this way we distinguish six steps.

In step (1) the Test Stand accepts as input an EXPERIMENT in the form of a tuple $\langle \mathcal{E}, \mathcal{D}, \mathcal{T}, \mathcal{Q}, \rangle$ where:

- \mathcal{E} is the RSP Engine subject of the evaluation (satisfying requirement [R.3]);
- \mathcal{D} is the input dataset [R.1];
- \mathcal{T} is the ontology [R.1];
- \mathcal{Q} is the query to be continuously answered by \mathcal{E} [R.2].

The TEST STAND executes the experiment $\langle \mathcal{E}, \mathcal{D}, \mathcal{T}, \mathcal{Q}, \rangle$ stressing \mathcal{E} for a certain period of time looping through the steps from (2) to (5) illustrated in Figure 1.1.

In step (2), the STREAMER pushes to \mathcal{E} an event CTEVENT. This event is a portion of an RDF Stream picked from the data \mathcal{D} and it consists of a set of RDF triples with the same timestamp. In order to satisfy [R.12], it sends triple in N-Triple¹, which is the easiest RDF serialisation to parse.

In step (3) \mathcal{E} pushes to the RESULT COLLECTOR an event OUTCTEVENT. It contains the current answer to the query \mathcal{Q} registered in \mathcal{E} given the ontology \mathcal{T} . The TEST STAND expects \mathcal{E} to output result in N-Triple format.

Notably, to place any RSP engine on the TEST STAND (requirement [R.3]) Heaven provides a simple software wrapper that, when it receives a CTEVENT, adapts it to the RSP engine specific format, pushes it in the RSP engine, and listens to the RSP engine output so to transform such an output in a OUTCTEVENT.

To measure performances (requirement [R.6]) the TEST STAND performs several actions both before step (2) and after step (3). To measure latency, it starts a timer before (2) and stops it after (3). To measure memory load, it asks for the free memory of the system after step (3). In step (4), those observations are added to the outputs of \mathcal{E} as annotations and are pushed to the RESULT COLLECTOR. We name TSRESULT the event that contains the sensor data plus the query results produced by the engine. The TEST STAND works in a single thread mode, blocking the execution of its components when it perform the performance measurements in (2) and (3) [R.4].

¹<http://www.w3.org/2001/sw/RDFCore/ntriples/>

In step (5) the RESULT COLLECTOR saves TSRESULT for post process analysis [R.9], executed through the ANALYZER. It does so appending TSResults to an open file to fulfil [R.8].

We define there main kinds of experiment, distinguishing on the data we want to sample and save. This distinction is relevant from an experimental point of view: indeed, ask the system for the memory usage may influence the latency calculus or saving on disk the query results may influence the memory footprint. From the Analyser point of view we have:

- Latency Experiment, where only the latency is calculated and no query result is saved on file
- Memory Experiment, where only the memory is gathered and no query result is saved on file
- Query Experiment, where query results are saved on file.
- Any combination of the previous experiment types.

The output of the Test Stand is designed with the Entity Relation model in figure 1.2. The diagram does not include entity attributes, but the relative Logic Schema, which follows, reports each entity field:

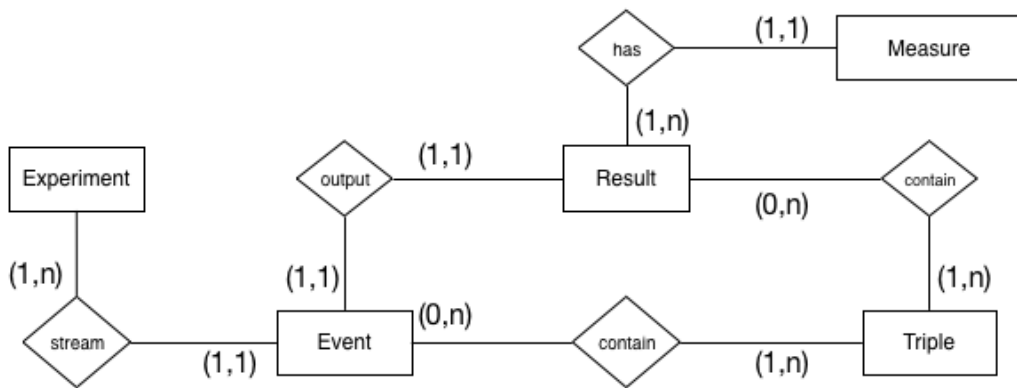


Figure 1.2: ER-Diagram For Experiment Output

Heaven

EXPERIMENT(ID, Timestamp Start, Timestamp End, Engine, Ontology, Query, Dataset, Description)

EVENT(ID, Experiment ID, Timestamp)

RESULT(Result ID, Experiment ID, Event ID)

MEASURE(ID, Value)

MEASUREMENT SET(Measure ID, Result ID, Experiment ID)

TRIPLE(S, P, O)

OUTPUT TRIPLE(Result ID, Experiment ID, S, P, O)

INPUT TRIPLE(Event ID, Experiment ID, S, P, O)

The EXPERIMENT entity contains all the metadata of the tuple $\langle \mathcal{E}, \mathcal{D}, \mathcal{T}, \mathcal{Q}, \rangle$. We also save when it start and when it finishes since can be relevant metrics for further analysis. Each EVENT is unique inside an EXPERIMENT, even in case of the identical triple set. Moreover, a RESULT EVENT is associated with one and only one EVENT. It contains the results to the engine queries w.r.t the current window and the measurements set gathered during the execution. The measurement set is extendible to fulfil requirement [R.7] In practice the Test Stand outputs results in time series format and the Analyser toolset has to handle this kind of information.

1.2 Baselines

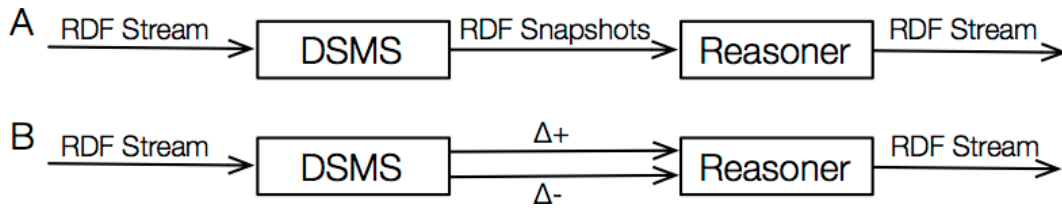


Figure 1.3: A: the architecture of the Naive baselines. B: the one of the Incremental ones.

In Chapter ?? we state that a Systematic Comparative Research Approach needs initial terms of comparison to lead the investigation. *Heaven* contains a set simple and easy-to-use RSP Engines called "baselines", developed to fulfil this lack. We exploit them to define some qualitative methods of investigation and to prove the usability of the Test Stand. In section ?? we identifies

four characteristics that classify a case-study as a baseline inside a research field. Indeed, in this section we present how *Heaven* Baselines are *Elementary*, *Relevant*, *Simple* and *Eligible*.

As explained in Section ??, the Stream Reasoning research area has three main building blocks: 1) RDF streams, 2) an extensions of SPARQL to manage continuous data and 3) reasoning algorithms. These three elements are summarised into RSP Engines, systems that can apply reasoning techniques upon rapidly changing information (Section ??). Early works on SR [3, 10] describe the most simple approach to create a stream reasoning system: pipelining a DSMS with a reasoner.

The first one, the DSMS, is responsible to handle the data stream, moving from infinite sequences to finite (and processable) sets of events. The second one, the reasoner, applies SPARQL queries on this set of events, exploiting its reasoning capabilities over a context that can be considered as static, but remains continuous.

Follow this approach does not require any new design effort. We just need to link two existing technologies and develop the communication between them. The proposed baselines are *Elementary* because they pipeline Esper², a mature and open source DSMS, with the Jena general purpose rule engine³ that is a flexible reasoning engine. Esper and Jena are two excellences in their context, Baseline *Eligibility* comes from their value. We can consider their couple as a simple but valid term of comparison.

Baselines *Relevance* means to cover all the most important theoretical variants the approach above conveys. In terms of reasoning and with reference to the data stream processing model, four baseline implementations cover two main design decisions about the RDF Stream Model and the Reasoning architecture. The first one, the RDF Stream model, describes how the input RDF Stream is processed, different systems accept data in different models, which depends on how RDF Stream is considered in terms of events contemporaneity. The two most relevant ones are:

- Triple-based model, where the events pushed in the DSMS are time-stamped triples. The timestamps are non decreasing, different triples could have the same timestamp to denote that they are contemporary.

²<http://esper.codehaus.org/>

³<http://jena.apache.org/documentation/inference/#rules>

- RDF Graphs-based: the event pushed in the DSMS are timestamped RDF graph. The timestamps are increasing and the graph is used as a form of punctuation [8] to separate consequent portions of the RDF stream.

The Reasoning Architecture, depends on the techniques to make inference, which are strictly related to the way data can flow from the DSMS to the reasoner. Again the possibility are two:

- Snapshot, where all the window contents is sent to the reasoner at each window slide
- IRStream, where the DSMS outputs the differences between the current window and the previous one

Thus, two possible solutions for reasoning on the triples in the window are possible:

- Naive solution: as shown in Figure 1.3-A, the DSMS determines the triples in the current window, produces an RDF snapshot of it and the reasoner materialise all the implied triples at each cycle. This approach is implemented in the C-SPARQL Engine [1] and in Sparkwave [5].
- Incremental solution: as shown in Figure 1.3-B, the DSMS outputs the IRStream. The Δ^+ snapshot contains the new triples that have just entered in the current window, while the Δ^- snapshot contains the triples that have just exited from the current window. The reasoner, using Δ^+ and Δ^- , incrementally maintains the materialisation over time. This approach is taken as term of comparison in [2] and it is inspired from [7].

The baseline answer the most simple query that involves a big reasoning effort: the identity query, which requires to materialise all the implicit information entailed by the content of the window, given the ontology chosen by the user. We chose ρ DF as entailment regime, because it is the minimal meaningful task for a Stream Reasoner. It is the minimal RDF-S fragment which reduce complexity but preserves the normative semantics and the core functionalities [6], several works in the field exploit it [9] and we do it as well to provide Simplicity.

Previous works on correctness of RDF stream processing [4] explain the importance of external control on time to assure that the RSP Engine always outputs the correct answers (even when overloaded). The proposed baselines take advantage of the ability of Esper to be temporally controlled by an external agent⁴ by sending time-keeping events to synchronise the internal time flow. One time-keeping event is sent before injecting the triples in a TCEVENT and another one after all triples in TCEVENT were sent. In this way all the triples in the TCEvent are consider contemporary by the baselines. Prove baselines soundness and completeness is important to sustain their Eligibility, and to allow the user to use them as an oracle for its own engine evaluation.

More consideration on Baseline Eligibility are presented in Chapter of ??, which deeply answer the questions about baselines performance with experimental results.

1.3 Analyser

The Analyser is conceptually a set of tools that allows visualisation and statistical investigation of data produced by executing an EXPERIMENT. As reported above (Section 1.1), the Test Stand result are time series, whose study is a very wide matter. To this extent, it is difficult, at this point of our research, designing a general architecture for the Analyser. Another motivation regards the impossibility of a complete decouple the analysis from the tool that can sustain it. Indeed, they both depend on the hypothesis of the research, which is deeply related to the experiment.

However, it is possible to identify some common characteristics that have a general meaning. Independently from the hypothesis and experiment we can consider many investigation methods, in particular the Systematic Comparative Approach as presented in Chapter ??), which requires to identify characteristics upon which is possible to build comparison.

First of all, the Analyser tool-set must include an automatic procedure to repeat the experiment execution and average the data. Even if the Test Stand is designed to be system independent, strange response may happen while an experiment is running, because it is a dynamic system too. In order

⁴http://esper.sourceforge.net/esper-0.7.5/doc/reference/en/html_single/index.html#api-controlling-time

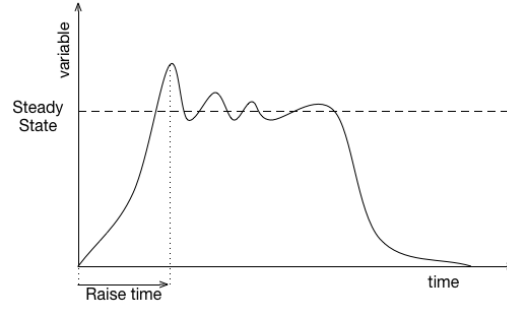


Figure 1.4: Time Series temporal domain

to reduce and possible eliminate the Byzantine errors, multiple runs of the same experiment must be mediated obtaining the average measures.

Once we have statistical relevant data, we can decline the comparative research approach either to the visual analysis or statistical investigation. The first method is more qualitative then the second one, but it is easier to read the information presented in graphical way. On the other hand, the statistical investigation method demands instruments to read the information in the data, once they are arranged in a certain form.

Time series studies usually start with data visualisation analysis. The most common plotting tools represent the series in temporal form. Moreover, it is relevant to visualise their behaviour in the frequency domain. The Steady State identification step is common in almost any research on dynamic system. We define Steady State the moment when the system reaches a dynamic equilibrium for a certain variable. Figure 1.4 shows how a typical dynamic system behaviour, for a specific variable, in the time domain. It also evidences the point when the system reaches the Steady State condition for that variable. Visual analysis can be exploited as the simplest tool for Steady State identification. This is useful to exclude the initial warm-up period form the data and properly verify hypothesis. Anyhow, it is a limited method, because different Steady States may happen for different system variables. This analysis requires to control the Steady State for each variable and to weight and explain their relation with further studies.

Independently from the Steady State condition, variables relation can be evidenced in a comparative form. The Analyser must support this kind of study. We distinguish, in the visual analysis context, three main comparative investigations. The following images show a some examples of this analysis:

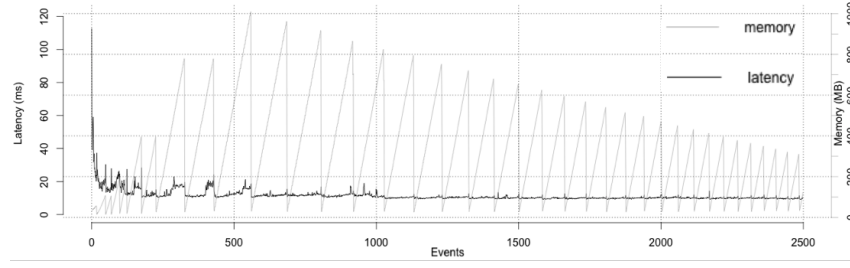


Figure 1.5: Intra Experiment Comparison

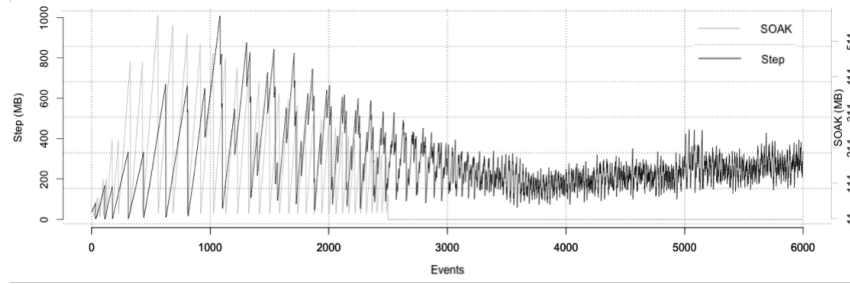


Figure 1.6: Inter Experiment Comparison

- A Figure 1.5 shows an example of intra-experiment investigation, multi-plotting them on the same domain;
- B Figure 1.6 shows an example of inter-experiment investigation: multi-plotting the same variable, arguing about the similarity or the difference upon a certain fixed characteristic.
- C Figure 1.7 shows an example of pattern recognition: inter-experiment on the same variable or intra-experiment on the relation between variable;

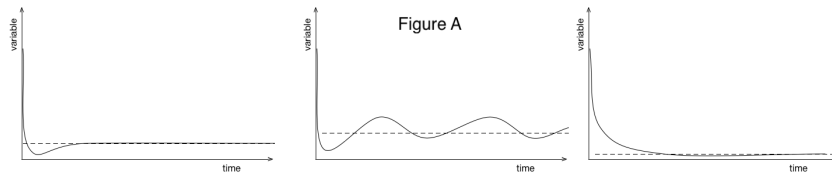


Figure 1.7: Pattern recognition example

The Analyser has to supports also hypothesis on system behaviour changes. A time serie describes how a dynamic system evolves over time, so it is meaningful to attempt hypothesis verification trough statistical operations, which consider the Steady State: average and standard deviation calculus, maximum and minimum identification for a certain variable. Upon this calculated data

is possible to compare experiments or variable in a global manner. What we did for the visual analysis comparison is almost still valid in this context, so we still distinguish between intra-experiment and inter-experiment comparisons, but we can chose to look at numeric values or to provide only an analysis of the system trend. Pattern identification is quite harder, but it depends on the data representation. The detail level of the investigation depends on the needs of the research:

(a) Symbolic Comparison of variables A vs B on Experiment 1					(b) Percentage Comparison of Experiments 1 vs 2 on variable A				
A vs B Comparison		Experiment 1 Condition A			E1A vs E2A Comparison		Experiment 1		
Experiment 1 Condition B		\simeq			Experiment 2	\simeq			
	A	\simeq				10%	\simeq		
	A	\simeq	\simeq			22%	\simeq	\simeq	
A	\simeq	B	B	32%		\simeq	-11%	-14%	

Table 1.1: (a) intra experiment-comparison over two variables - (b) inter-experiment comparison over a common variable

Last but not least, since data mining procedures are very system-dependent we decide to include examples of possible analysis in Chapter ?? about the experimental evaluation of the Baselines. The purpose of Chapter ?? is both to demonstrate the value of *Heaven* and to provide some guidelines for further evaluations.

Bibliography

- [1] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. Querying RDF streams with C-SPARQL. *SIGMOD Record*, 39(1):20–26, 2010.
- [2] Emanuele Della Valle Daniele Dell’Aglio. Incremental reasoning on rdf streams. In Andreas Harth, Katja Hose, and Ralf Schenkel, editors, *Linked Data Management*, chapter 16, pages 413–436. CRC Press, 2014.
- [3] Emanuele Della Valle, Stefano Ceri, Davide Francesco Barbieri, Daniele Braga, and Alessandro Campi. A first step towards stream reasoning. In *FIS*, pages 72–81, 2008.
- [4] Daniele Dell’Aglio, Jean-Paul Calbimonte, Marco Balduini, Óscar Corcho, and Emanuele Della Valle. On correctness in RDF stream processor benchmarking. In *ISWC*, pages 326–342, 2013.
- [5] Srdjan Komazec, Davide Cerri, and Dieter Fensel. Sparkwave: continuous schema-enhanced pattern matching over RDF data streams. In *DEBS*, pages 58–68, 2012.
- [6] Sergio Muñoz, Jorge Pérez, and Claudio Gutiérrez. Minimal deductive systems for RDF. In *ESWC*, pages 53–67, 2007.
- [7] Yuan Ren and Jeff Z. Pan. Optimising ontology stream reasoning with truth maintenance system. In *CIKM*, pages 831–836, 2011.
- [8] N. Tatbul, U. Cetintemel, S. B. Zdonik, M. Cherniak, and M. Stonebraker. Exploiting punctuation semantics in continuous data streams. *IEEE Trans. on Knowledge and Data Eng.*, 15(3):555–568, 2003.

BIBLIOGRAPHY

- [9] Jacopo Urbani, Alessandro Margara, Criel J. H. Jacobs, Frank van Harmelen, and Henri E. Bal. Dynamite: Parallel materialization of dynamic RDF data. In *ISWC Part I*, pages 657–672, 2013.
- [10] Onkar Walavalkar, Anupam Joshi, Tim Finin, and Yelena Yesha. Streaming knowledge bases. In *In International Workshop on Scalable Semantic Web Knowledge Base Systems*, 2008.