Publication of RDF Streams with Ztreamy

Jesús Arias Fisteus(✉), Norberto Fernández García,
Luis Sánchez Fernández, and Damaris Fuentes-Lorenzo

Dpto. Ing. Telemática, Universidad Carlos III de Madrid,
Avda. Universidad 30, 28911 Leganés, Madrid, Spain
{jaf,berto,luiss,dfuentes}@it.uc3m.es

Abstract. There is currently an interest in the Semantic Web community for the development of tools and techniques to process RDF streams. Implementing an effective RDF stream processing system requires to address several aspects including stream generation, querying, reasoning, etc. In this work we focus on one of them: the distribution of RDF streams through the Web. In order to address this issue, we have developed Ztreamy, a scalable middleware which allows to publish and consume RDF streams through HTTP. The goal of this demo is to show the functionality of Ztreamy in two different scenarios with actual, heterogeneous streaming data.

1 Introduction

Nowadays there are many practical applications (like social or sensor networks) where the information to be processed takes the form of a stream, that is, *a real-time, continuous, ordered, sequence of items* [5], where each item describes an application event (social network post, sensor measurement, etc.).

The popularization of streaming data applications has fostered the interest of the Semantic Web community for this kind of data. As a result of this interest, a W3C community group on RDF Stream Processing[1] has recently started.

There are several aspects that need to be taken into account when implementing an effective RDF stream processing system. For instance, querying the RDF streams (with proposals like C-SPARQL [2] and $\text{SPARQL}_{\text{Stream}}$ [3]), reasoning on streams [9], scaling the stream processing [7], etc.

Another aspect to be taken into account is the scalable publication of RDF streams on the Web. To address this issue, we have developed Ztreamy [1], a middleware for large-scale distribution of RDF streams on top of HTTP. As shown in [1], Ztreamy, is able to publish a real-time stream to tens of thousands of simultaneous clients with delays of a few seconds, outperforming in this aspect the mechanisms for stream publication integrated in alternative platforms such as DataTurbine [8] and Linked Stream Middleware [6]. To achieve these results, Ztreamy relies on the use of buffering strategies, stream compression and a single-threaded non-blocking input/output paradigm at the server.

[1] http://www.w3.org/community/rsp/ (March 12th, 2014)

V. Presutti et al. (Eds.): ESWC Satellite Events 2014, LNCS 8798, pp. 286–291, 2014.
DOI: 10.1007/978-3-319-11955-7_36

```
Event-Id: a360bdd8-695b-4e5b-b74e-bdaaec3eeafe
Source-Id: wikipedia-changes-002
Application-Id: wikipedia-changes
Syntax: text/n3
Timestamp: 2014-03-12T18:30:58+01:00
Body-Length: 296

@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix webtlab: <http://webtlab.it.uc3m.es/> .

webtlab:_642529540 dc:date "2014-03-12T17:30:28Z" ;
   webtlab:pageid "16866376" ;
   webtlab:title "Persona (satellite)" .
```

Fig. 1. An example of item in a Ztreamy stream.

The main goal of this demo is to show the functionality of Ztreamy. In particular, we demonstrate how applications can publish their own RDF streams and how these streams can be consumed by other applications. The demonstration involves actual, heterogeneous streaming data coming from two different sources: (i) physical sensors placed in a room; and, (ii) information obtained by monitoring the activities of Wikipedia editors.

2 A Brief Introduction to Ztreamy

Ztreamy is a scalable middleware platform for the distribution of RDF streams on the Web. Its main objective is serving RDF streams on top of HTTP to as many clients as possible with minimal delays.

A stream in Ztreamy is a sequence of data items, where each item is composed by a RDF graph and metadata about it, such as its creation timestamp and the identifier of its source entity. Figure 1 shows an example of a data item.

Stream servers are the core of Ztreamy. They aggregate the items they receive from data producers (e.g. physical sensors) into streams, and provide those streams to consumers with a publish-subscribe paradigm. The platform is flexible with respect to how streams are served an manipulated. For example, a stream can easily be filtered, split into separate streams, joined with other streams in order to form aggregate streams, etc. Streams can be served from just one server or replicated from many servers.

Like the Web, the Ztreamy platform is intended to be open. It allows any entity to publish its own streams just by installing a server. It is also possible to mirror any stream that can be consumed or publish derivative streams.

Data producers and consumers communicate with Ztreamy servers through HTTP. Therefore, producers and consumers can be programmed with almost any programming language and run on almost any platform. This includes fully-fledged servers for stream processing, desktop applications, JavaScript applications that run on Web browsers, mobile applications and embedded systems.

Ideally, consumers subscribe to a stream with long-lived HTTP requests. With this mechanism the server sends the response in chunks as new data is

available, but never finishes the response neither closes the connection. This way of communication is efficient because just one underlying TCP connection is used and just one HTTP request needs to be processed for a possibly long period of time. Some HTTP client libraries and HTTP proxies may not be compatible with long-lived requests, as they expect responses to be complete before retransmitting them. In that case, consumers can use long-polling instead, in which the server finishes the response as soon as all the available data has been sent, and the consumer sends a new HTTP request immediately after that.

The Ztreamy platform provides some basic built-in services ready to use by applications, such as some simple semantics-based filters. However, it does not aim at being a complete platform for RDF stream processing. Applications that need more complex services such as a stream query engine can integrate them on top of Ztreamy, or use other systems for the data processing tasks and leave Ztreamy just for publishing the streams.

The main focus of our work was on scalable publication of streams to large amounts of simultaneous consumers. Therefore, we carried out a performance evaluation of Ztreamy in which we show that it outperforms other existing solutions in that task [1]. Our experiments show that the main factors that contribute to achieving that level of performance are:

- Buffering data at the server: Instead of sending new data to consumers as soon as it is available, it is buffered and sent periodically. The experiments show that even with very small periods (e.g. 0.5 s), which barely delay the delivery of data, there are big gains in the use of CPU of the servers. Thus, the servers are able to handle more simultaneous clients.
- Compressing the streams: Streams in Ztreamy can be compressed with the Deflate stream compression protocol. In combination with the server buffering mechanism, our experiments show not only a reduction of about 85 % in network traffic but also additional gains in the use of CPU in the server, which allows it to handle more clients.
- Use of single-threaded non-blocking input/output: Ztreamy servers are built on top of the Tornado Web server, which was designed to handle large amounts of simultaneous clients. To do so, Tornado uses a single-threaded non-blocking input/output paradigm based on the new asynchronous facilities of modern operating systems. Many servers are recently switching from the multi-threaded to the non-blocking paradigm because of its performance advantages when handling many simultaneous network connections.

We have published the current prototype of Ztreamy, which is implemented with the Python programming language, under the free software GNU GPL license. More information about the platform is available at [1] and at its website[2].

[2] http://www.it.uc3m.es/jaf/ztreamy (March 12th, 2014)

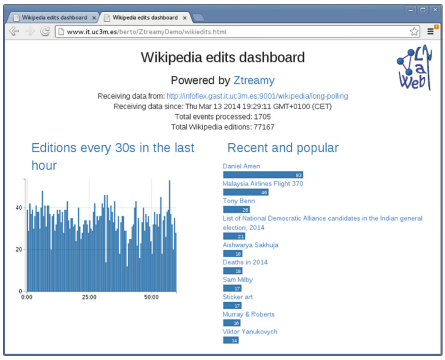

Fig. 2. Wikipedia edits demo interface.

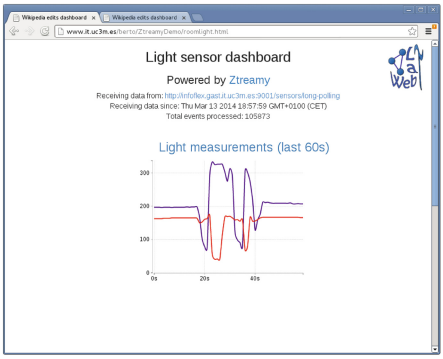

Fig. 3. Light sensor demo interface.

3 Ztreamy Demo

The main goal of the demo is to show how applications can publish their own RDF streams using Ztreamy and how can these streams be consumed by other applications. In particular, we center our demo in two different scenarios:

- **Wikipedia edits**: In this scenario, a Python application acts as information source. It uses the Wikipedia API[3] to monitor the activity of Wikipedia editors. Every 30 s, this applications generates a new data item (similar to that in Fig. 1) with metadata about the edits recently carried out in the encyclopedia (including the timestamp and the title of the modified page). Then, using HTTP, the application sends the item to a Ztreamy stream server that publishes it in a stream. A second application connects to this server to consume the stream. It is a Web application[4] implemented in JavaScript that runs in the Web browser of the user. It uses the *ztreamy.js* library in order to interact with the stream server. The interface of the application (depicted in Fig. 2) includes two graphs: one graph shows the number of edits every 30 s in the last hour, and a second graph shows a ranking with the top 10 Wikipedia pages by number of edits (from a list containing the last 2000 edited pages). Both graphs are dynamic: they are updated every time a new item in the stream is received at the browser.
- **Physical sensors**: In this scenario two information sources publish items to the same stream. In particular, each information source consists of a PC connected to an Arduino and a TSL235R light sensor. Every second, a Python application installed in each of the PCs, reads the measurements provided by the local sensor through an USB port and generates a Ztreamy stream item. Then, it publishes the item into a single stream server shared between the

[3] http://en.wikipedia.org/w/api.php (March 12th, 2014)
[4] Available at: http://www.it.uc3m.es/berto/ZtreamyDemo/wikiedits.html

different sources. The stream server integrates the information coming from the two PCs into a single stream that is consumed by a Web application[5]. The consumer application shows in a single graph (see Fig. 3) the evolution in real-time of the measurements provided by the sensors.

4 Conclusions and Future Lines

This demo has shown how applications can take advantage of the functionalities provided by Ztreamy to publish RDF streams or to consume RDF streams published by third parties. We have also shown how the information provided by different sources can be easily integrated into a single stream. Though we have used Python to implement the information providers and JavaScript to implement the consumers, given that Ztreamy relies on HTTP as communication protocol, it is possible to use other alternatives. Thus, implementing libraries to interact with Ztreamy from other languages, like Java or Ruby, is a work to be developed in the near future. At the moment Ztreamy relies on the general-purpose Zlib stream compressor, which implements Deflate. However, we are considering as a future line the possibility of integrating RDF stream compression algorithms like [4] into the system.

Acknowledgements. This work has been partially funded by the Spanish Government through the project HERMES-SMARTDRIVER (TIN2013-46801-C4-2-R).

References

1. Arias, J., Fernández, N., Sánchez, L., Fuentes-Lorenzo, D.: Ztreamy: a middleware for publishing semantic streams on the Web. Web Semant. Sci. Serv. Agents World Wide Web **25**, 16–23 (2014). doi:10.1016/j.websem.2013.11.002
2. Barbieri, D.F., Braga, D., Ceri, S., Grossniklaus, M.: An execution environment for C-SPARQL queries. In: Proceedings of the 13th International Conference on Extending Database Technology, EDBT '10, pp. 441–452 (2010)
3. Calbimonte, J.-P., Corcho, O., Gray, A.J.G.: Enabling ontology-based access to streaming data sources. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 96–111. Springer, Heidelberg (2010)
4. Fernández, N., Arias, J., Sánchez, L., Fuentes-Lorenzo, D., Corcho, Ó.: RDSZ: an approach for lossless RDF stream compression. In: Presutti, V., d'Amato, C., Gandon, F., d'Aquin, M., Staab, S., Tordai, A. (eds.) ESWC 2014. LNCS, vol. 8465, pp. 52–67. Springer, Heidelberg (2014)
5. Golab, L., Özsu, M.T.: Issues in data stream management. SIGMOD Rec. **32**, 5–14 (2003)
6. Le-Phuoc, D., Nguyen-Mau, H.Q., Parreira, J.X., Hauswirth, M.: A middleware framework for scalable management of linked streams. Web Semant. Sci. Serv. Agents World Wide Web **16**(5), 42–51 (2012)

[5] Available at: http://www.it.uc3m.es/berto/ZtreamyDemo/roomlight.html

7. Le-Phuoc, D., Nguyen Mau Quoc, H., Le Van, C., Hauswirth, M.: Elastic and scalable processing of linked stream data in the cloud. In: Alani, H., et al. (eds.) ISWC 2013, Part I. LNCS, vol. 8218, pp. 280–297. Springer, Heidelberg (2013)
8. Tilak, S., Hubbard, P., Miller, M., Fountain, T.: The ring buffer network bus (RBNB) dataturbine streaming data middleware for environmental observing systems. In: IEEE International Conference on e-Science and Grid Computing, pp. 125–133, December 2007
9. Valle, E.D., Ceri, S., Harmelen, Fv, Fensel, D.: It's a streaming world! reasoning upon rapidly changing information. IEEE Intell. Syst. **24**(6), 83–89 (2009)