Ulysses: an Intelligent client for replicated Triple Pattern Fragments

Thomas Minier[1], Hala Skaf-Molli[1], Pascal Molli[1], and Maria-Ester Vidal[2]

[1] LS2N, University of Nantes, Nantes, France
firstname.lastname@univ-nantes.fr
[2] TIB Leibniz Information Centre For Science and Technology
University Library & Fraunhofer IAIS, Germany
Maria.Vidal@tib.eu

Abstract. Ulysses is an intelligent TPF client that takes advantage of replicated datasets to distribute the load of SPARQL query processing and provides fault-tolerance. By reducing the load on a TPF server, Ulysses improves the Linked Data availability and distributes the financial costs of queries execution among data providers. This demonstration presents the Ulysses web client and shows how users can run SPARQL queries in their browsers against TPF servers hosting replicated data. It also provides various visualizations that show in real-time how Ulysses performs the actual load distribution and adapts to network conditions during SPARQL query processing.

Keywords: Semantic Web, Triple Pattern Fragments, Intelligent client, Load balancing, Fault tolerance, Data Replication

1 Introduction

We proposed Ulysses [1], a replication-aware intelligent TPF client that distributes the load of SPARQL query processing across heterogeneous replicated TPF servers. Ulysses relies on a light-weighted cost-model for computing servers processing capabilities and a client-side load balancer to distribute SPARQL query processing and provides fault tolerance during query processing.

Consider the SPARQL query Q_1 in Figure 1, and the two servers S_1 and S_2 publishing a replica of the DBpedia 2015 dataset, hosted by DBpedia [3] and LANL Linked Data Archive [4], respectively. Executing Q_1 with the regular TPF client [4] on S_1 alone generates **442 HTTP calls**, takes **7s** in average, and returns 222 results. Executing the same query as a federated SPARQL query on both S_1 and S_2 generates **478 HTTP calls** on S_1 and **470 HTTP calls** on S_2, returns 222 results, and takes **25s** in average. This is because *existing TPF clients do not support replication nor client-side load balancing* [1].

As Ulysses is aware that datasets hosted at S_1 and S_2 are replicated, it only generates **442 HTTP calls** that are distributed between servers according to

[3] http://fragments.dbpedia.org/
[4] http://fragments.mementodepot.org/

their processing capabilities and network latencies. If the servers are not loaded, the performances of Ulysses are similar to those of the regular TPF client ($7s$) without replication. However, if the servers are loaded, Ulysses improves significantly the performances thanks to load-balancing.

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?software ?company WHERE {
  ?software dbo:developer ?company. # tp1
  ?company dbo:locationCountry ?country . # tp2
  ?country rdfs:label "France"@en . # tp3
}
```

Fig. 1: SPARQL query Q_1 that finds all softwares developed by French compagnies

Using replicated servers, Ulysses prevents a single point of failure server-side, improves the overall availability of data, and distributes the financial costs of queries execution among data providers.

This demonstration presents the Ulysses web client. It details which informations are collected by Ulysses about servers in real-time, how the cost model is recomputed, and how the load of SPARQL query processing is balanced among replicated servers through different real-time visualizations. Finally, Ulysses reactions in presence of servers failure are illustrated.

2 Overview of Ulysses client

The Ulysses web client is available online at `http://ulysses-demo.herokuapp.com`. In order to distribute the load of SPARQL query processing across heterogeneous TPF servers hosting replicated data, it relies on three key ideas detailed in [1]. In next sections, we provides a brief overview of key ideas and how they are integrated in the Ulysses web client[5].

2.1 Replication-aware source selection

Ulysses uses a *replication-aware source selection* algorithm to identify which TPF servers can be used to distribute evaluation of triple patterns during SPARQL query processing, based on the replication model introduced in [2,3].

This replication model allows to describe replicated datasets using *replicated fragment* and a *fragment mapping*. A fragment is defined as 2-tuple : the authoritative source of the fragment, and a triple pattern met by the fragment's triple. A fragment mapping is a function that maps each fragment to a set of TPF

[5] The open-source Ulysses client is available at `https://github.com/Callidon/ulysses-tpf`, under MIT license.

servers. Using these information, ULYSSES is able to compute relevant sources for all triple pattern in a SPARQL query.

Consider again the two servers S_1, S_2 and the SPARQL query Q_1 in Figure 1. Only one fragment $f_1 = \langle$ `http://fragments.dbpedia.org/2015-10/en`, `?s ?p ?o`$\rangle$ is defined to indicate a total replication. A fragment mapping $\mathcal{F}$ maps f_1 to the set $\{S_1, S_2\}$. Thus, all RDF triples met by every triple pattern of Q_1 are replicated by both DBpedia and LANL servers.

For simplicity, in this demonstration we only consider the scenario with total replication. Consequently, the evaluation a triple pattern of the query Q_1 will be distributed between servers DBpedia and LANL.

2.2 A cost-model for estimating servers processing capabilities

ULYSSES uses response times of HTTP requests performed against TPF servers during query processing as probes to accurately estimate the *processing capabilities* of a server. The response time of each request is used to compute the throughput of a server, *i.e.*, the number of results server per unit of time by a server. As SPARQL query processing with the TPF approach requires to send many requests to a server in order to evaluate triple patterns, ULYSSES can keep the servers throughputs updated in real-time without additional probing. This can also easily detect load spikes or server failures.

Servers' throughputs are used to compute a *cost-model* that define a *capability factor* of each TPF server. This capability factor determines the load distribution among servers: a server with a high capability factor has more chance to be selected to evaluate a triple pattern as detailed in Section 2.3.

Real-time statistics

Estimated load

Server	Access time	Page size	Throughput	Server capability factor	Estimated load
http://fragments.dbpedia.org/2015-10/en	139ms	100 triples	0.714 triple/ms	1	25%
http://fragments.mementodepot.org/dbpedia_201510	264ms	500 triples	2.392 triple/ms	3	75%

Fig. 2: ULYSSES cost-model, updated in real-time

Figure 2 shows a real-time estimation of servers loads during execution of query Q_1 of Figure 1 against S_1 and S_2. S_1 is slightly faster to access than S_2, but as the latter serves five times more results per access (**Page size** column), S_2 has a better throughput than S_1. As, S_2 has a better capability factor than S_1, it will receive approximately 75% of the query load, while S_1 will approximately receive the remaining 25% (**Estimated load** column)

2.3 Adaptive client-side load balancing with fault tolerance

Ulysses uses an *adaptive load-balancer* to perform load balancing among replicated servers. Each evaluation of a triple pattern scheduled by the client is sent to a server selected using a weighted random algorithm, inspired by the Smart clients approach [5]. The probability of selecting a server is proportional to its processing capabilities, according to Ulysses cost-model.

This probability distribution ensures that each TPF server will only process an amount of requests proportional to its processing capabilities, without concentrating all the load of query processing on the most performant servers. Ulysses load-balancer also provides *fault-tolerance*, by re-scheduling failed HTTP requests using available replicated servers.

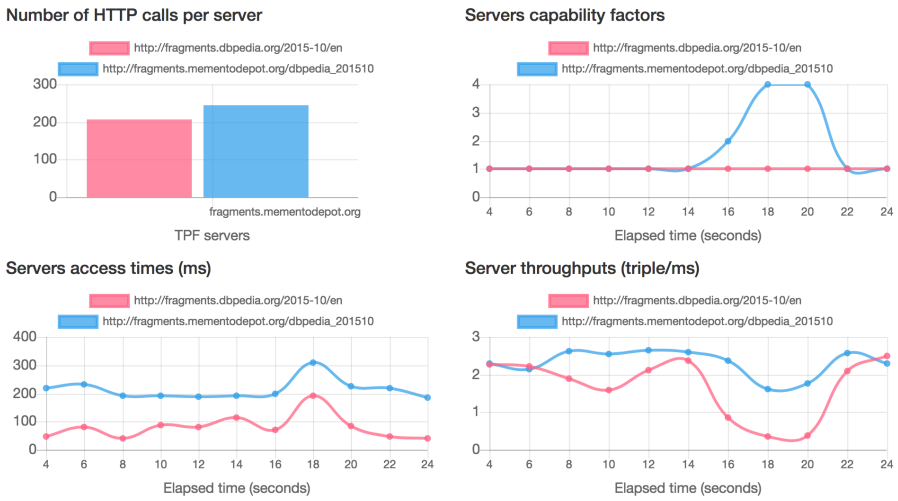

Fig. 3: Metrics recorded by Ulysses and used to perform load-balancing during SPARQL query processing

Figure 3 shows the metrics displayed in real-time by the Ulysses web client during SPARQL query processing of Q_1, distributed among S_1 and S_2. We see that the server throughputs and capability factors of both servers remain close at the start of query processing (**Server access times** and **Servers capability factors**). However, after 18 seconds, S_1 access times increase, so S_2 became more efficient than S_1, causing its capability factor to rise. Thus, the load distribution is affected in real-time, and, at the end of query processing, we see that S_2 has received more HTTP requests (**Number of HTTP requests per server**).

3 Demonstration scenario

In the context of ESWC 2018, we would like to run a live experiment that anyone can join. We will tweet a link that participants can click to access Ulysses online demonstration, using their laptops or smartphones. Then, they will be able to submit SPARQL queries against a set of TPF servers hosting replicated data. We will provide a selection of replicated TPF servers, hosting replicas of DBpedia and WatDiv datasets, with some SPARQL queries as a quick-start. Participants will also be able to use their own set of TPF servers and SPARQL queries.

In this scenario, participants will be able to see how Ulysses keeps its cost-model updated in real-time and how it benefits of this to distribute the load of query processing, using visualizations presented in Figure 2 and Figure 3. Additionally, we will also provide replicated TPF servers that can be shutdown in order to simulate failures. Participants will be able to see how Ulysses is able to continue query processing after a server failure, by re-distributing the load using available servers.

4 Conclusion

In this demonstration, we presented the Ulysses web client that enables Web Browsers to perform client-side load balancing and provides fault tolerance when evaluating SPARQL queries against TPF servers hosting replicated data. Real-time visualizations allow to observe how Ulysses distributes the load of SPARQL query processing across replicated TPF servers according to their processing capabilities, and adapts to failures or variations in network conditions.

Acknowledgments This work is partially supported through the FaBuLA project, part of the AtlanSTIC 2020 program.

References

1. Minier, T., Skaf-Molli, H., Molli, P., Vidal, M.: Intelligent clients for replicated triple pattern fragments. In: Proceedings of the 15th Extended Semantic Web Conference (ESWC 2018) (2018)
2. Montoya, G., Skaf-Molli, H., Molli, P., Vidal, M.E.: Federated sparql queries processing with replicated fragments. In: International Semantic Web Conference. pp. 36–51. Springer (2015)
3. Montoya, G., Skaf-Molli, H., Molli, P., Vidal, M.E.: Decomposing federated queries in presence of replicated fragments. Web Semantics: Science, Services and Agents on the World Wide Web 42, 1–18 (2017)
4. Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., De Meester, B., Haesendonck, G., Colpaert, P.: Triple pattern fragments: A low-cost knowledge graph interface for the web. Web Semantics: Science, Services and Agents on the World Wide Web 37, 184–206 (2016)
5. Yoshikawa, C., Chun, B., Eastham, P., Vahdat, A., Anderson, T., Culler, D.: Using smart clients to build scalable services. In: Proceedings of the 1997 USENIX Technical Conference. p. 105. CA (1997)