Ladda: SPARQL Queries in the Fog of Browsers

Arnaud Grall[1], Pauline Folz[1,2], Gabriela Montoya[3], Hala Skaf-Molli[1], Pascal Molli[1], Miel Vander Sande[4], and Ruben Verborgh[4]

[1] LS2N – Nantes University, France
{arnaud.grall@etu., firstname.lastname@}univ-nantes.fr
[2] Nantes Métropole – Research, Innovation and Graduate Education Department, France
[3] Department of Computer Science – Aalborg University, Denmark
gmontoya@cs.aau.dk
[4] Ghent University – imec – IDLab, Belgium {miel.vandersande,ruben.verborgh}@ugent.be

Abstract. Clients of Triple Pattern Fragments (TPF) interfaces demonstrate how a SPARQL query engine can run within a browser and re-balance the load from the server to the clients. Imagine connecting these browsers using a browser-to-browser connection, sharing bandwidth and CPU. This builds a fog of browsers where end-user devices collaborate to process SPARQL queries over TPF servers. In this demo, we present Ladda: a framework for query execution in a fog of browsers. Thanks to client-side inter-query parallelism, Ladda reduces the makespan of the workload and improves the overall throughput of the system.

1 Introduction

Clients of Triple Pattern Fragments (TPF) interfaces demonstrate how a SPARQL query engine can run within a browser and re-balance the load from the server to the clients [4].

However, executing a workload composed of many queries with a single browser has intrinsic limitations regarding CPU and bandwidth. These limitations are severe as TPF potentially generates many calls and high network traffic.

Imagine connecting TPF clients using a browser-to-browser connection, sharing bandwith and CPU. This realizes a *fog of browsers* [3] in which decentralized end-user devices cooperate to process SPARQL queries.

In this demo, we present Ladda; a framework for query execution in a fog of browsers. Ladda bypasses previous limitations and improves system throughput through the concurrent execution on multiple processors. Such *inter-query parallelism* was traditionally realized on the server-side. Ladda enables inter-query parallelism on the client-side, significantly reduces the time to obtain all results (*makespan*) and improves the overall *throughput*.

2 Ladda: query delegation in the fog

Continuing previous work [1], we introduce the Ladda framework for query execution in a fog of browsers. A federation of data consumers is connected through a Random Peer Sampling (RPS) overlay network [5]. Such a network approximates a random graph where each data consumer is connected to a fixed number of neighbors. It is resilient to churn, to failures and communication with neighbors is a zero-hop.

In the context of browsers, basic communications rely on WebRTC[5] to establish a data-channel between browsers and SPRAY [2] to enable RPS on WebRTC. Each browser maintains a set of neighbors K called a *view* that is a random subset of the whole network. To keep its view random, a data consumer renews it periodically by shuffling its view with the view of a random neighbor.

A browser executes an infinite stream of queries that arrive at any time. A data consumer can *execute* its query or *delegate* it to a neighbor. Given a fog of browsers and a workload

[5] https://webrtc.org/

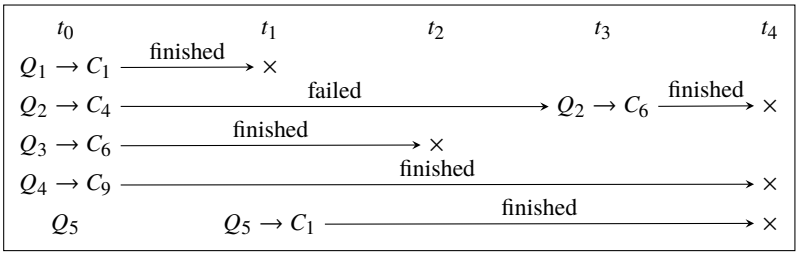

Fig. 1: Execution of C_1's workload of five queries with three neighbors among ten clients.

of queries distributed in time across browsers, we aim to minimize the results time for data consumers, where, Δ, is the time elapsed between query results time ($Q.rt$) and query arrival time ($Q.at$). Ladda implements a load-balancing algorithm to balance the load among neighbors by executing queries on *free* neighbors.

Consider a federation of ten data consumers C_1 to C_{10}, where each data consumer has three neighbors, *i.e.*, $size(K) = 3$. Consider the data consumer C_1 has a workload of five queries, $C_1.W = [Q_1, ..., Q_5]$ and the following neighbors: C_4, C_6 and C_9.

Figure 1 illustrates how Ladda executes the workload of C_1. At time t_0, C_1 allocates its queries as follows: $(Q_1 \rightarrow C_1)$, $(Q_2 \rightarrow C_4)$, $(Q_3 \rightarrow C_6)$, and $(Q_4 \rightarrow C_9)$. Consequently, C_4, C_6 and C_9 belong to the list of *busy* neighbors of C_1: $C_1.B = [C_4, C_6, C_9]$.

- At time t_1, C_1 has finished the execution of Q_1, C_1 becomes *free* and it has only one waiting query, $C_1.W = [Q_5]$, therefore, C_1 executes Q_5: $(Q_5 \rightarrow C_1)$.
- At time t_2, Q_3 finished. As all queries are allocated, there is nothing to do.
- At time t_3, Q_2 delegation fails. C_6 is no longer *busy* for C_1, so we allocate Q_2 to C_6.
- At time t_4, Q_2, Q_4 and Q_5 are finishing.

3 Evaluation

We evaluated Ladda on a local TPF server providing the DBpedia 3.8 dataset with the HDT back-end and four workers, a Web cache and different numbers of clients. NGINX is configured as a Web cache with a size of 1GB. The TPF server, the Web cache, and all the TPF clients run on the same machine: a HPC server with 40 processors, 130 GB of memory, and Debian 7.8. From the DBpedia 3.8 query log, we extracted a full hour of queries from 50 clients (1,509 queries in total) on one day. We considered two setups as follows.

All loaded: 50 clients have their own query workload. This is considered as the worst case for Ladda, because at the beginning all clients are *busy*, so the first delegations to neighbors always fail.

One loaded: One client has the full workload. Since the client is the only one *busy* in the federation, all delegations succeed.

Figure 2 shows how Ladda improves the makespan for the two configurations.

4 Online demo

The Ladda online demo is available at `https://ladda-demo.herokuapp.com/`, and the source code is available at `https://github.com/folkvir/ladda-demo`.

When visiting `https://ladda-demo.herokuapp.com/`, the browser downloads and starts Ladda locally. Ladda connects the browser with other browsers in order to build a set of neighbors. Ladda needs to know at least one connected participant. A signaling service

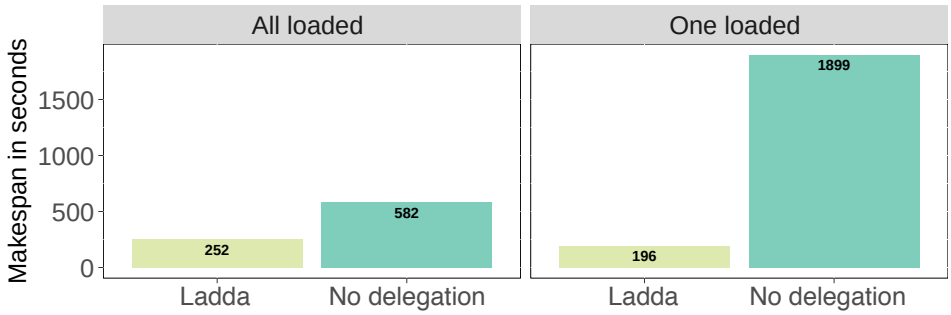

Fig. 2: Ladda delegation significantly decreases the makespan, both when each of the 50 clients has its own workload (worst case) and in the case where one client has the entire workload (best case).

running on `https://ladda-demo.herokuapp.com/` facilitates this process by keeping a random subset of connected browsers.

The number of neighbors "#Neighbors" appears in the timeline panel (see Figure 3). Thanks to SPRAY [2], this number is bounded to $log(N)$ where N is the number of connected browsers.

Once connected, a browser can delegate queries to neighbors. The query workload appears in the "Queries" panel. Queries are executed against the TPF server defined in the "Queries panel". The delegation number dn determines the number of simultaneous delegation tentative. For instance, for a workload of 5 queries $q1$-$q5$, 3 neighbors and $dn = 2$, Ladda executes locally $q1$, delegates $q2$ and $q3$ to random neighbors and terminates its allocation process. A new allocation process takes place on the next event: receiving results, delegation failed or timeout on a delegation.

The button "Execute" allows to launch the execution of the local workload. The "Timeline" panel and the "Query" panel display in real-time the progression of the execution.

- The timeline allows to know in real-time when a participant executes a query. Each line represents a participant including "me". On a participant c, a query q starts at time $q.st$ and terminates at time $q.et$. The timeline displays for each participant $q.st$ and $q.et$.
- The "Query" panel displys the status of queries : waiting, done or delegated.

When the workload execution is terminated, Ladda computes the following statistics:

Global execution time is the makespan's workload. We suppose all queries in the workload arrive at the same time fixed when the user click on the "execute" button. So the global execution time is the difference between the result time of the last executed query and the arrival time.

Sequential execution time. For all Q_i in the workload, the sequential execution time is $\sum_{Q_i} Q_i.et - Q_i.st$.

Improvement ratio is the ratio for the global execution time to the sequential one. This comparison is barely an approximation because we cannot ensure that queries execution time would be the same if queries are executed sequentially.

Overhead is the total transfer time of queries and results between the browser and neighbors.

Once connected, it is also possible to receive delegated queries. The "Remote Queries" link in the timeline allows to display received queries.

5 Ladda demo scenario

In the context of ESWC 2017, we would like to run a live experiment that any ESWC 2017 participant can join. We will start the replay of the DBpedia logs available in USEWOD 2016

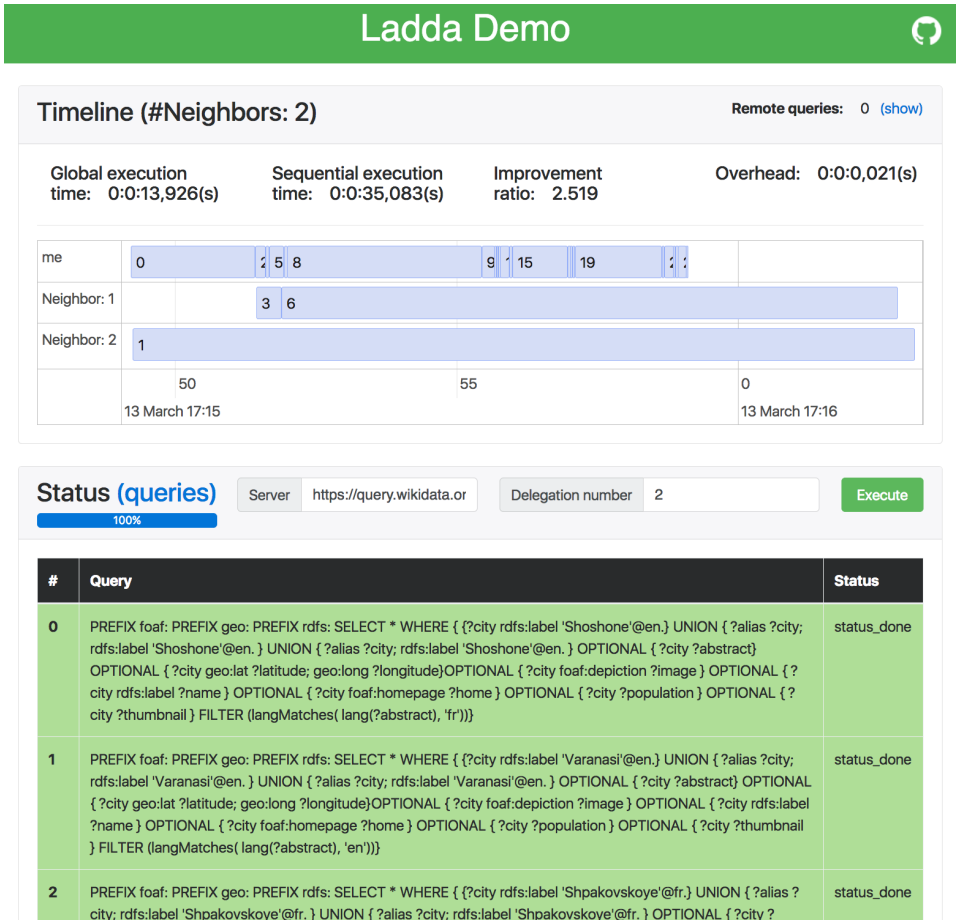

Fig. 3: Ladda interface after executing queries.

with TPF [6]. DBpedia logs contain hundreds of thousands of queries that a single browser can hardly execute. We will evaluate the throughput of a monitored TPF server according to the number of participants. We expect to see that collaboration between participants allows to increase the throughput of the system.

During the conference, we will tweet a link that anyone with a compatible browser (Chrome and Firefox) can click on and join the experiment. Participants will be able to see which queries they execute and observe in real-time the throughput of the system.

We aim to confirm that the number of participants positively impacts the throughput, *i.e.*, great improvements on the throughput are observed when the number of users increases.

6 Conclusion and future work

In this paper, we presented Ladda, an approach to execute SPARQL queries in the fog of browsers. Ladda enables inter-query parallelism on the client side. Ladda significantly reduces the overall makespan and improves the throughput of the federation.

In this demo, we did not take into accounts latencies in the network, neighbors are chosen randomly. A first perspective is to take into account network latencies to choose neighbors

Second, It is interesting to study how collaborative caching as provided by Cyclades [1] and inter-query parallelism provided by Ladda contribute to performances improvements.

[6] If compatible with the USEWOD usage agreement, otherwise, we will use synthetic queries.

Finally, in Ladda, we focused on inter-query parallelism. Another research direction is to consider intra-query parallelism. Decomposition of SPARQL queries and delegation of subqueries open interesting perspectives.

Acknowledgement

We thank Thibaud Courtoison, Maël Quémard and Sylvain Vuylsteke, students of the Computer Science Department at the University of Nantes for implementing the interface of Ladda.

References

1. P. Folz, H. Skaf-Molli, and P. Molli. CyCLaDEs: a decentralized cache for Linked Data Fragments. In *ESWC: Extended Semantic Web Conference*, 2016.
2. B. Nédelec, P. Molli, and A. Mostefaoui. Crate: Writing stories together with our browsers. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 231–234. International World Wide Web Conferences Steering Committee, 2016.
3. L. M. Vaquero and L. Rodero-Merino. Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, 2014.
4. R. Verborgh, M. Vander Sande, O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Haesendonck, and P. Colpaert. Triple Pattern Fragments: a low-cost knowledge graph interface for the Web. *Journal of Web Semantics*, 37–38:184–206, Mar. 2016.
5. S. Voulgaris, D. Gavidia, and M. Van Steen. CYCLON: inexpensive membership management for unstructured P2P overlays. *Journal of Network and Systems Management*, 13(2):197–217, 2005.