Computing Continuous SPARQL Query over RDF Streams on Storm Platform

By Usman Younas and Sander Breukink

# Introduction

The amount of data on the web has massively increased over the years, and it doesn’t show any signs of stopping. Hence the demand for analyzing it in reasonable amount of time is also increasing. The Resource Description Framework (RDF) is a data model whose purpose is to form a comprehensive framework to integrate data from different fields. It is a flexible data model used in the Semantic Web (a Web of data) on which we can do querying or reasoning. In this report we discuss several ways of executing SPARQL queries on RDF data and compare their results based on several testing scenarios. For this we will use the Apache Storm Framework with different topologies. We will also run these topologies on a testing set and compare these results.

# Decomposing queries

The first part of our project consisted of decomposing the SPARQL queries on the RDF data. An RDF triple consists of a subject, a predicate and an object. The query decomposition means turning a selection query in SPARQL into Java code executed on RDF triples, note that we only focus on selection queries, in other words: we just look at the triples, we don’t change them.

We handle this tuple by tuple, so we loop through all the tuples and keep the ones that satisfy the query. We have 3 categories of queries:

* One variable joins: there’s only one input string, this string has to match the subject of the triples, the object and predicate can be anything.
* Two variable joins: There are 2 input strings, one has to match with the subject and one has to match with the object, if both of these match, the triple is passed on.
* Multi variable joins: There are 3 input strings. These strings are compared with the all 3 variables in the triple, if 2 of the input variables match with 2 of the variables in the triple, the triple is passed on.

RDF triples that match the query go on for further analysis, all the other triples are dropped and we no longer look at them.

# Spouts

In the RDF framework, we can create several spouts. In our program, a spout fetches the data and passes it on to a bolt, which we’ll discuss after this section. In our topologies, we choose to use only a single tuple every time because we have only one data source.

The spout calls a function nextTuple() every 100 milliseconds. This functions makes sure a new line from the input file get read, takes the data from this line and transforms it into a rdf triple which get’s passed on to the collector that passes it on to the bolts.

For more information on the architecture, we refer to the documentation on apache storm which can be found at [1].

# Bolts

Every time the spout emits a triple, it get’s passed on to the first bolt which is the next class that handles the triples. What the bolt does depends on the topology, we’ll discuss the different topologies in the next section.

We can have multiple bolts running in parallel, in most of our cases we have 3 bolts running in parallel. Bolts also handle the data and then pass them on. After that the outputted data is either the end result or is handled by another bolt.

# Topologies

A topology manages all the spouts and bolts, it also contains the main() functions from which everything starts. Each of our topologies has it’s own characteristics. We’ll discuss all of them here.

## IVJTopology

Firstly we obtain the tuples from the spout functions that reads the txt file and converts it’s contents into RDF triples. Those triples get divided over 3 bolts that run in parallel, each bolt has 2 bloomfilters, bloom filters are explained at [2]. Each bolt creates a bloom filter and stores the predicate of the tuple in the bloom filter. Based on whether it’s a one variable join, a two variable join or a multi variable join, it get’s added to these bloom filters. Then these bloom filters get sent on to the next bolt, the next bolt, of which there’s just one, takes these bloom filters and changes them into actual results.

## Count Based Topology

Todo

## Grid based topology

In the grid based topology we have 3 spouts running in parallel, one outputting all the work tuples, one for all the Diploma tuples and one for all the paper tuples. Each of these spouts emits a different subset of all the tuples.

After that we also launch 3 bolts, again for papers, work and diplomas. Each of the bolts checks of the tuples match with the query or not, if they do, the tuple gets added to the bloom filter which is used to obtain the final results.

## Time based topology

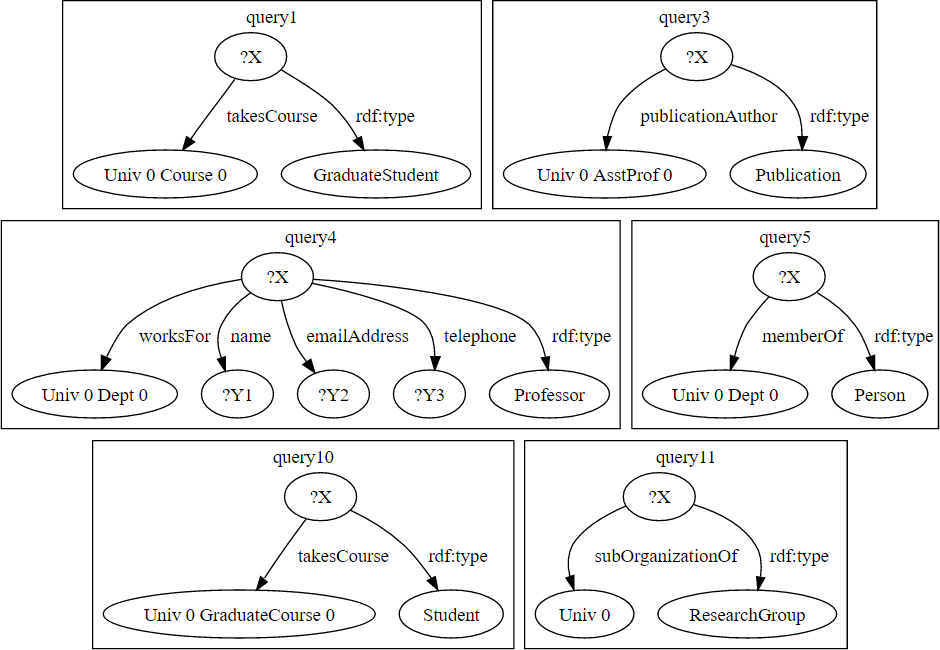
In the time based topology all tuples are outputted by a single spout. They then get passed on to one of the 3 builder bolts who all pass their data on to the one prober bolt. The bolts try to analyze a new tuple every 100ms. In case no new tuple has been emitted before the bolt tries to analyze, the message “Timer is exited’ get’s printed.

## Topology with Three Bloom Filters

Todo

# Testing scenarios

For testing the performance of our different topologies, we generated a data set and performed queries from the Lehigh University Benchmark, an overview of this benchmark facility can be found at [3]. The code allowed us to generate 15 files with RDF data on a virtual university with professors, students, courses, publications, etcetera. We can perform several queries on these data sets, in total there are 14 possible queries, but for our benchmark test 6 of these where sufficient. A graphical representation of the six queries we used is shown in figure 1.



Figure

# Testing environment

Grid 5000 network

TODO

# Benchmark results

Results from the benchmarks

TODO

# Conclusion

TODO

# References

1. <http://storm.apache.org/releases/2.0.0-SNAPSHOT/index.html>
2. <https://en.wikipedia.org/wiki/Bloom_filter>
3. <http://swat.cse.lehigh.edu/projects/lubm/>