ParlBench: A SPARQL Benchmark for Electronic Publishing Applications[*]

Tatiana Tarasova and Maarten Marx

ISLA, University of Amsterdam, Science Park 904, 1098 XH Amsterdam, The Netherlands
{t.tarasova,maartenmarx@}uva.nl

Abstract. ParlBench is an RDF benchmark modelling a large scale electronic publishing scenario. The benchmark offers large collections of the Dutch parliamentary proceedings together with information about members of the parliament and political parties. The data is real, but free of intellectual property rights issues. On top of the benchmark data sets several application benchmarks as well as targeted micro benchmarks can be developed. This paper describes the benchmark data sets and 19 analytical queries covering a wide range of SPARQL constructs. The potential use of ParlBench is demonstrated by executing the queries for 8 different scaling of the benchmark data sets on Virtuoso RDF store. Measured on a standard laptop, data loading times varied from 43 seconds (for 1% of the data set) to 48 minutes (for the complete data set), and execution of the complete set of queries (570 queries in total) varied from 9 minutes to 13 hours.

Keywords: SPARQL, RDF benchmark, parliamentary proceedings.

1 Introduction

RDF stores are the backbones of RDF data driven applications. There is a wide range of RDF store systems available[1] together with various benchmark systems[2] to assess performances of the RDF stores.

As discussed in the Benchmark Handbook [1], different applications impose different requirements to a system, and the performance of the system may vary from one application domain to another. This creates the need for domain specific benchmarks. The existing application benchmarks for RDF store systems often employ techniques developed by the Transaction Processing Performance Council (TPC)[3] for relational databases and use synthetically generated data sets for their workloads. However, performance characteristics for loading and querying such data may differ from those that were measured on real life data sets, as it was shown by the DBpedia benchmark [2] on DBpedia [6]. To the best of our knowledge, among the existing benchmarks for RDF store systems, only the DBpedia benchmark provides a real data set.

[*] This work was supported by the EU- FP7 (FP7/2007-2013) project ENVRI (grant number 283465).

[1] http://www.w3.org/wiki/LargeTripleStores, last accessed: July 9, 2013.

[2] http://www.w3.org/wiki/RdfStoreBenchmarking, last accessed: July 9, 2013.

[3] http://www.tpc.org/, last accessed: July 10, 2013.

P. Cimiano et al. (Eds.): ESWC 2013, LNCS 7955, pp. 5–21, 2013.

With this work we propose the ParlBench application benchmark that closely mimics a real-life scenario: large scale electronic publishing with OLAP-type queries. ParlBench consists of (1) real life data and (2) a set of analytical queries developed on top of these data.

The benchmark data sets include the Dutch parliamentary proceedings, political parties and politicians. The ParlBench data fit very well the desiderata of Gerhard Weikum's recent Sigmod blog[4]: it is open, big, real, useful, linked to other data sources, mixing data-values and free text, and comes with a number of real-life workloads.

The queries in the benchmark can be viewed as coming from one of two use cases: create a report or perform a scientific research. As an example of the latter, consider the question whether the performance of males and females differs in parliament, and how that has changed over the years. To enable more comprehensive analysis of the RDF stores' performances, we grouped the benchmark queries into four micro benchmarks [3] with respect to their analytical aims *Average*, *Count*, *Factual* and *Top 10.*

The paper is organized as follows. Section 2 gives an overview of the related work. Section 3 describes the benchmark data sets. In Section 4 we define the benchmark queries and present the micro benchmarks. An experimental run of ParlBench on the Virtuoso RDF store is discussed in Section 5.

2 Related Work

There is a number of RDF store benchmarks available. The most relevant benchmarks to our work are discussed further. The Berlin SPARQL Benchmark (BSBM) [4] implements an e-commerce application scenario. Similarly to ParlBench, BSBM employed the techniques developed by the Transaction Processing Performance Council (TPC)[5], such as query permutations (for the Business Intelligence use case) and system ramp-up.

The SPARQL Performance Benchmark (SP^{2}Bench) [5] is settled in the DBLP scenario. SP^{2}Bench queries are carefully designed to test the behavior of RDF stores in relation to common SPARQL constructs, different operator constellations and RDF access patterns. SP^{2}Bench measures query response time in cold runs settings, i.e., when query execution time is measured immediately after the server was started.

Both the Berlin and SP^{2}Bench use synthetically generated data sets, whereas, the DBpedia SPARQL Benchmark (DBPSB) [2] uses a real data set, DBpedia [6]. In addition to using a real data set, the DBPSB benchmark uses real queries that were issued by humans and applications against DBpedia. These queries cover most of the SPARQL features and enable comprehensive analysis of RDF stores' performances on a single feature as well as combinations of features. The main difference between the ParlBench and DBPSB benchmarks is that the latter is not developed with a particular application in mind. Thus, it is more useful for a general assessment of the performance of different RDF stores' implementations, while ParlBench is particularly targeted on developers of e-publishing applications and can support them in choosing systems that are more suitable for analytical query processing.

[4] `http://wp.sigmod.org/?p=786`, last accessed: July 1, 2013.

[5] `http://www.tpc.org/`, last accessed: July 1, 2013.

3 Benchmark Data Sets

The benchmark consists of five conceptually separate data sets summarized in Table 1:

Members : describes political players of the Dutch parliament.
Parties : describes Dutch political parties.
Proceedings : describes the structure of the Dutch parliamentary proceedings.
Paragraphs : contains triples linking paragraphs to their content.
Tagged entities : contains triples linking paragraphs to DBpedia entities indicating that these entities were discussed in the paragraphs.

Table 1. Statistics of the benchmark data sets

data set	**# of triples**	**size**	**# of files**
members	33,885	14M	3,583
parties	510	612K	151
proceedings	36,503,688	4.15G	51,233
paragraphs	11,250,295	5.77G	51,233
tagged entities	34,449,033	2.57G	34,755
TOTAL:	**82,237,411**	**∼13G**	**140,955**

The data model of the benchmark data sets is described in Appendix A.

3.1 Scaling of the Benchmark Data Sets

The size of the ParlBench data sets can be changed in different ways. The data set can be scaled by the number of included proceedings. All proceeding files are ordered chronologically by year. In order to construct a scaled proceeding data set, we randomly sampled from each year of proceedings an equal number of proceedings. This was done to ensure uniform and sufficient presence of proceedings from each year. Optionally, one can include *Tagged entities* and/or *Paragraphs* data sets to the test collection. In this case *Paragraphs* and *Tagged Entities* are scaled accordingly to the included proceedings, i.e., only paragraphs and/or tags pointing to id's in the chosen proceedings are included.

4 Benchmark Queries

ParlBench provides 19 SPARQL queries. The queries were grouped into four micro benchmarks:

Average: 3 queries, numbered from `A0` to `A2`, retrieve aggregated information.
Count: 5 queries, numbered from `C0` to `C4`, count entities that satisfy certain filtering conditions.
Factual: 6 queries, numbered from `F0` to `F5`, retrieve instances of a particular class that satisfy certain filtering conditions.
Top 10: 5 queries, numbered from `T0` to `T4`, retrieve the top 10 instances of a particular class that satisfy certain filtering conditions.

All the queries are listed in Appendix B. Their SPARQL representations can be seen in Appendix C. The benchmark queries cover a wide range of the SPARQL language constructs. Table 2 shows the usage of SPARQL features by individual query and distribution of the features across micro benchmarks.

Table 2. SPARQL characteristics of the benchmark queries

micro benchmark	Average			Count					Factual						Top 10				
	A0	**A1**	**A2**	**C0**	**C1**	**C2**	**C3**	**C4**	**F0**	**F1**	**F2**	**F3**	**F4**	**F5**	**T0**	**T1**	**T2**	**T3**	**T4**
FILTER						+	+	+	+	+	+	+		+					
UNION	+	+	+					+		+	+	+					+		+
LIMIT										+			+		+	+	+	+	+
ORDER BY										+			+		+	+	+	+	+
GROUP BY	+	+	+				+	+		+			+		+	+	+	+	+
COUNT	+	+	+	+	+	+	+	+		+	+	+	+		+	+	+	+	+
DISTINCT									+			+	+	+					
AVG	+	+	+																
negation														+					
OPTIONAL									+					+					
subquery	+	+	+					+			+	+	+						
blank node scoping				+	+	+	+	+	+	+	+	+							
# of triple patterns	10	9	12	5	5	5	6	13	8	16	6	6	2	4	2	4	9	3	11

5 Experimental Run of the Benchmark

In this section we demonstrate the application of our benchmark on the Virtuoso RDF native store (OSE, v.06.01.3127)[6]. Tested on the Berlin benchmark, Virtuoso showed one of the best performance results among other systems [4].

5.1 Experimental Setup

Test Environment. For the benchmark experiment we used a personal laptop Apple MacBook Pro with Intel i7 CPU (2x2 cores) running at 2.8 GHz and 8GB memory. See Appendix D for a more detailed specification of the test environment.

Evaluation Metrics

Loading Time. The loading time is the time for loading RDF data into an RDF store. The benchmark data sets are in RDF/XML format. The time is measured in seconds. Loading of data into Virtuoso was done one data set at a time. For the loading of *Parties* and *Members* we used the Virtuoso RDF bulk load procedure. For *Proceedings* we used the Virtuoso function `DB.DBA.RDF_LOAD_RDFXML_MT` to load large RDF/XML text.

Query Response Time. The query response time is the time it takes to execute a SPARQL query. To run the queries programmatically, we used `isql`, the Virtuoso interactive SQL utility. The execution time of a single query was taken as the real time returned by the bash `/usr/bin/time` command. 10 permutations of the benchmark queries were created, each containing 19 SPARQL queries.

Before starting measuring the query response time, we *warmed-up* Virtuoso by running 5 times 10 different permutations of all 19 queries of the benchmark. In total, 950 queries were executed in the *warm-up* phase, and each query was run 50 times. After that, we run the same permutations 3 more times and measured the execution time of each query. The query response time was computed as the mean response time of executing each query 30 times.

Test Collections. Experiments are run on 8 test collections. Each collection includes the *Parties* and *Members* data sets and a scaled *Proceedings* data set ranging from 1 to 100% . Table 3 gives an overview of the size of each test collection.

[6] `http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/`, last accessed: July 1, 2013.

Table 3. Sizes of the test collections for different scaling of *Proceedings*

Scaling Factor	1%	2%	4%	8%	16%	32%	64%	100%
# of triples	494,875	1,027,395	1,906,880	3,851,642	7,554,304	15,129,621	23,341,602	36,542,431

5.2 Results

We report on three experiments, relating database size to execution time: (1) time needed to load the test collections (see fig. 1), (2) total time needed to execute all the queries in micro benchmarks[7] (see fig. 2), and (3) query execution time of all the queries on the largest collection (see fig. 3).

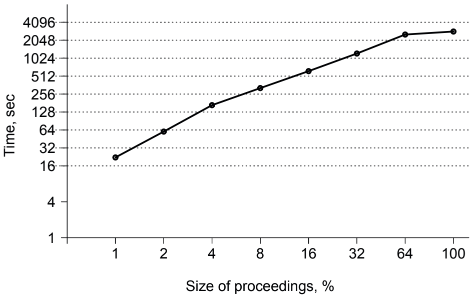

Fig. 1. Loading Time in sec of the benchmark collections

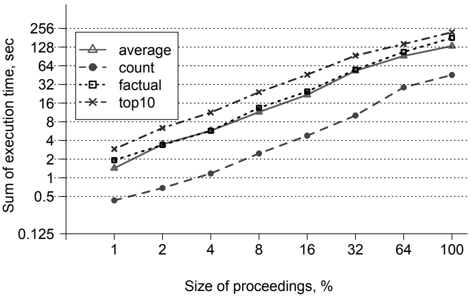

Fig. 2. Query Execution Time in sec of micro benchmarks on the test collections

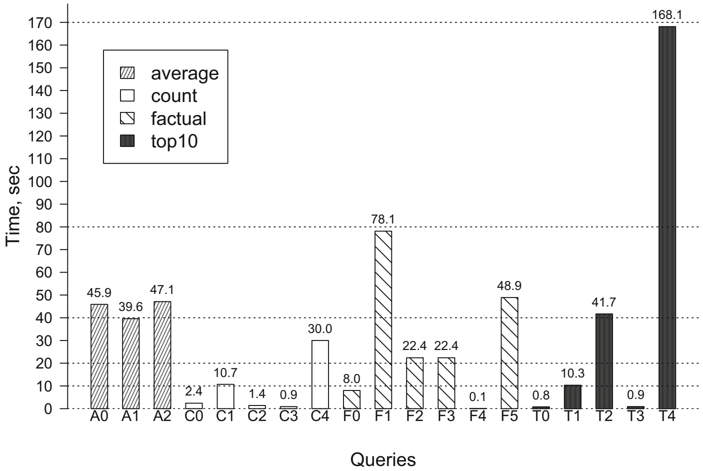

Fig. 3. Query Execution Time in sec of the benchmark queries on the largest test collection

The y-axes on fig. 1 and fig. 2 are presented in a log scale, and the numbers represent the loading and query response time in seconds. To make the results reproducible, we publish the benchmark data sets, queries and scripts at `http://data.politicalmashup.nl/RDF/`.

[7] For each group we summed the execution time of each query in the group.

6 Conclusion

ParlBench has the proper characteristics of an RDF benchmark: it can be scaled easily and it has a set of intuitive queries which measure different aspects of the SPARQL engine.

We believe that ParlBench is a good proxy for a realistic large scale digital publishing application. ParlBench provides real data that encompass major characteristics shared by most of the e-publishing use cases including rich metadata and hierarichal organization of the content into text chunks.

The data set is large enough to perform non-trivial experiments. In addition to the analytical scenario presented, one can think of several other application scenarios that can be developed on the same data sets. Due to the many and strong connections of the benchmark to the Linked Open Data Cloud through the DBpedia links, natural Linked Data integration scenarios can be developed from ParlBench. The ParlBench data is also freely available in XML format [7], enabling cross-platform comparisons of the same workload.

As a future work we will consider the execution of the benchmark on other RDF stores and comparison of the results with the ones achieved on Virtuoso.

Another interesting direction for future work could be to extend the set of queries. Currently, there are only two queries with the `OPTIONAL` operator and one query with `negation`. Queries that use these and other features of SPARQL 1.1 could be a good addition to the benchmark. ParlBench has many queries that extensively use the `UNION` to construct various paths based on the `hasPart` property. We could re-write these queries through the SPARQL 1.1. path expressions and exploit the transitive property of `hasPart` to test the reasoning capabilities of RDF store systems.

References

1. Gray, J.: The Benchmark Handbook for Database and Transaction Systems, 2nd edn. Morgan Kaufmann (1993)
2. Morsey, M., Lehmann, J., Auer, S., Ngonga Ngomo, A.-C.: DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 454–469. Springer, Heidelberg (2011)
3. Afanasiev, L., Manolescu, I., Michiels, P.: MemBeR: A Micro-benchmark Repository for XQuery. In: Bressan, S., Ceri, S., Hunt, E., Ives, Z.G., Bellahsène, Z., Rys, M., Unland, R. (eds.) XSym 2005. LNCS, vol. 3671, pp. 144–161. Springer, Heidelberg (2005)
4. Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. International Journal on Semantic Web and Information Systems 5(2), 1–24 (2009)
5. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP^{2}Bench: A SPARQL Performance Benchmark. In: ICDE, pp. 222–233. IEEE (2009)
6. Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: DBpedia - a crystallization point for the web of data. Journal of Web Semantics (JWS) 7, 154–165 (2009)
7. Maarten, M.: Advanced Information Access to Parliamentary Debates. Journal of Digital Information (JoDI) 10(6) (2009)
8. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of SPARQL. ACM Transactions on Database Systems (TODS) 34(3), 16:1–16:45 (2009)

A Data Model

Figure 4 gives an overview of the data model. The data model consists of the following classes:

- `PoliticalParty` represents political parties. Parties are linked to their equivalent resources in DBpedia.
- `ParliamentMember` represents politicians, i.e., members of the parliament. Members are linked to the DBpedia resources that correspond to the same politicians. Additionally, members have biographical information such as gender, birthday and the place of birth, and the death date and place if applicable.
- `ParliamentaryProceedings` are written records of parliamentary meetings.
- `Topic` represents a different point on the agenda in the proceedings.
- `Scene` and `StageDirection` are important structural elements of the Dutch parliamentary proceedings. They contain information about current speakers and their interrupters.
- `Speech` is a constituent of a *Topic*, a *Scene* or *Stage Direction*. *Speech* represents a beginning of a speech of a new speaker.
- `Paragraph` is a container for all spoken text; can be part of any structural element of the proceedings described above.

The structural elements of the proceedings are connected to `Proceedings` through the `dcterms:hasPart` property. The speaker of the speech and the affiliated party of the speaker are attached to `Speech` via the `refMember` and `refParty` properties correspondingly.

Vocabularies. Relevant existing vocabularies and ontologies to represent documents of parliamentary proceedings fall into two categories. In the first category we identified vocabularies that are too generic, such as the SALT Document Ontology[8] or the DoCo, the Document Components Ontology[9]. They do not provide means to represent such specific concepts as *stage direction* or *scene*. Vocabularies in the second category are too specific. For example, the Semantic Web Conference Ontology[10] or the Semantic Web Portal Ontology[11] define classes and properties to model conference proceedings.

[8] `http://salt.semanticauthoring.org/ontologies/sdo#`

[9] `http://purl.org/spar/doco/Paragraph`

[10] `http://data.semanticweb.org/ns/swc/swc_2009-05-09.html#`

[11] `http://sw-portal.deri.org/ontologies/swportal#`

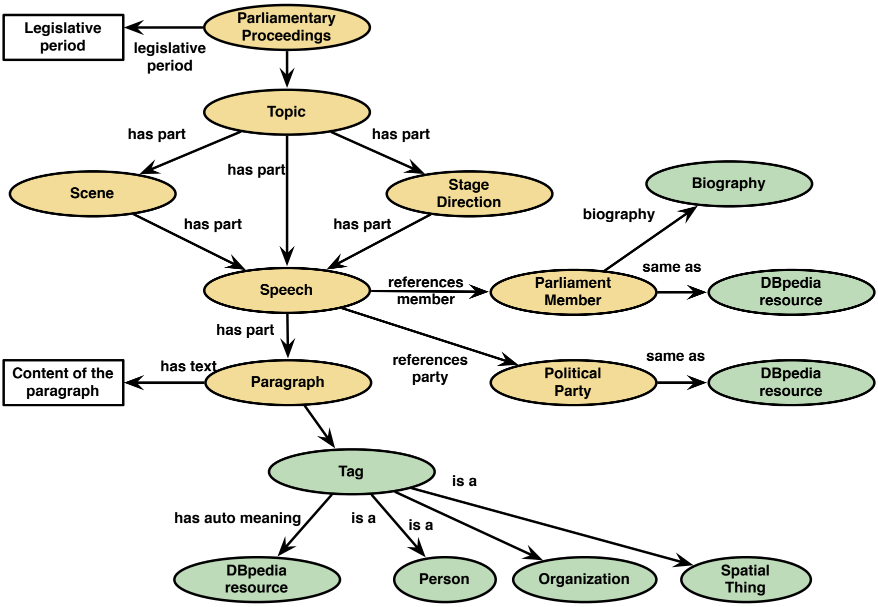

Fig. 4. Data model of the benchmark data sets

We defined our own RDFS vocabulary to model parliamentary proceedings, the Parliamentary Proceedings vocabulary[12], and integrated it with other existing vocabularies. To represent biographical information of politicians, we used BIO: A vocabulary for biographical information[13] together with the Friend of a Friend Vocabulary (FOAF)[14] and the DBpedia Ontology[15]. The Modular Unified Tagging Ontology (MUTO)[16] was used to represent information about tagged entities of paragraphs. The Dublin Core Metadata Terms[17] was used to encode metadata information.

[12] `http://purl.org/vocab/parlipro#`

[13] `http://vocab.org/bio`

[14] `http://xmlns.com/foaf/0.1/`

[15] `http://dbpedia.org/ontology/`

[16] `http://muto.socialtagging.org/`

[17] `http://purl.org/dc/terms/` and `http://purl.org/dc/elements/1.1/`

B ParlBench Queries

Table 4. List of the benchmark queries

		average
1.	A0	Retrieve average number of people spoke per topic.
2.	A1	Retrieve average number of speeches per topic.
3.	A2	Retrieve average number of speeches per day.
		count
4.	C0	Count speeches of females.
5.	C1	Count speeches of males.
6.	C2	Count speeches of speakers who were born after 1960.
7.	C3	Count speeches of male speakers who were born after 1960.
8.	C4	Count speeches of a female speaker from the topic where only one female spoke.
		factual
9.	F0	What members were born after 1950, their parties and dates of death if exist?
10.	F1	What is the gender of politicians who spoke most between 1995 and 2005?
11.	F2	What is the percentage of male speakers?
12.	F3	What is the percentage of female speakers?
13.	F4	What politician has most number of Wikipedia pages in different languages?
14.	F5	What speeches are made by politicians without Wikipedia pages?
		top 10
15.	T0	Retrieve top 10 members with the most number of speeches.
16.	T1	Retrieve top 10 topics when most of the people spoke.
17.	T2	Retrieve top 10 topics with the most number of speeches.
18.	T3	Retrieve top 10 days with the most number of topics.
19.	T4	Retrieve top 10 longest topics (i.e., topics with the most number of paragraphs).

C ParlBench SPARQL Queries

Table 5. Prefixes used in the SPARQL queries

rdf:	`<http://www.w3.org/1999/02/22-rdf-syntax-ns#>`
parlipro:	`<http://purl.org/vocab/parlipro#>`
dcterms:	`<http://purl.org/dc/terms/>`
dc:	`<http://purl.org/dc/elements/1.1/>`
bio:	`<http://purl.org/vocab/bio/0.1/>`
foaf:	`<http://xmlns.com/foaf/0.1/>`
dbpedia:	`<http://dbpedia.org/resource/>`
owl:	`<http://www.w3.org/2002/07/owl#>`

Table 6. SPARQL representation of the benchmark queries

A0: Retrieve average number of people spoke per topic.

```
SELECT AVG(?numOfMembers) as ?avgNumOfMembersPerTopic
WHERE {{
SELECT COUNT(?member) AS ?numOfMembers
WHERE {
  ?topic rdf:type parlipro:Topic .
  ?speech rdf:type parlipro:Speech .
  ?speech parlipro:refMember ?member .
  {?topic dcterms:hasPart ?speech .}
UNION{
  ?topic dcterms:hasPart ?sd .
  ?sd rdf:type parlipro:StageDirection .
  ?sd dcterms:hasPart ?speech .}
UNION{
  ?topic dcterms:hasPart ?scene .
  ?scene rdf:type parlipro:Scene .
  ?scene dcterms:hasPart ?speech .}}
GROUP BY ?topic}}
```

A1: Retrieve average number of speeches per topic.

```
SELECT AVG(?numOfSpeeches) as ?avgNumOfSpeechesPerTopic
WHERE {{
SELECT COUNT(?speech) AS ?numOfSpeeches
WHERE {
  ?topic rdf:type parlipro:Topic .
  ?speech rdf:type parlipro:Speech .
  {?topic dcterms:hasPart ?speech .}
UNION{
  ?topic dcterms:hasPart ?sd .
  ?sd rdf:type parlipro:StageDirection .
  ?sd dcterms:hasPart ?speech .}
UNION{
  ?topic dcterms:hasPart ?scene .
  ?scene rdf:type parlipro:Scene .
  ?scene dcterms:hasPart ?speech .}}
GROUP BY ?topic}}
```

A2: Retrieve average number of speeches per day.

```
SELECT AVG(?numOfSpeeches) as ?avgNumOfSpeechesPerDay
WHERE {{
SELECT ?date COUNT(?speech) AS ?numOfSpeeches
WHERE {
  ?proc dcterms:hasPart ?topic .
  ?proc rdf:type parlipro:ParliamentaryProceedings .
  ?proc dc:date ?date .
  ?speech rdf:type parlipro:Speech .
  ?topic rdf:type parlipro:Topic .
  {?topic dcterms:hasPart ?speech .}
UNION{
  ?topic dcterms:hasPart ?sd .
  ?sd rdf:type parlipro:StageDirection .
  ?sd dcterms:hasPart ?speech .}
UNION{
  ?topic dcterms:hasPart ?scene .
  ?scene rdf:type parlipro:Scene .
  ?scene dcterms:hasPart ?speech .}}
GROUP BY ?date}}
```

C0: Count speeches of females.

```
SELECT COUNT(?speech)
WHERE {
   ?speech rdf:type parlipro:Speech .
   ?speech parlipro:refMember ?member .
   ?member bio:biography _:bio .
   _:bio rdf:type bio:Biography .
   _:bio foaf:gender dbpedia:Female .}
```

C1: Count speeches of males.

```
SELECT COUNT(?speech)
WHERE {
   ?speech rdf:type parlipro:Speech .
   ?speech parlipro:refMember ?member .
   ?member bio:biography _:bio .
   _:bio rdf:type bio:Biography .
   _:bio foaf:gender dbpedia:Male .}
```

C2: Count speeches of speakers who were born after 1960.

```
SELECT COUNT(?speech)
WHERE {
  ?speech rdf:type parlipro:Speech .
  ?speech parlipro:refMember ?member .
  ?member bio:biography _:bio .
  _:bio rdf:type bio:Biography .
  _:bio foaf:birthday ?birthday .
FILTER (year(?birthday) > 1960)}
```

C3: Count speeches of male speakers who were born after 1960.

```
SELECT COUNT(?speech)
WHERE {
  ?speech rdf:type parlipro:Speech .
  ?speech parlipro:refMember ?member .
  ?member bio:biography _:bio .
  _:bio rdf:type bio:Biography .
  _:bio foaf:gender dbpedia:Male .
  _:bio foaf:birthday ?birthday .
FILTER (year(?birthday) > 1960)}
```

C4: Count speeches of a female speaker from the topic where only one female spoke.

```
SELECT ?topic ?member COUNT(?speech) as ?numOfSpeeches
WHERE {{
SELECT ?topic ?member COUNT(?member) AS ?numOfFemales ?speech
WHERE {
  ?topic rdf:type parlipro:Topic .
  ?speech rdf:type parlipro:Speech .
  ?speech parlipro:refMember ?member .
  ?member bio:biography _:bio .
  _:bio rdf:type bio:Biography .
  _:bio foaf:gender dbpedia:Female .
  {?topic dcterms:hasPart ?speech .}
UNION{
  ?topic dcterms:hasPart ?sd .
  ?sd rdf:type parlipro:StageDirection .
  ?sd dcterms:hasPart ?speech .}
UNION{
  ?topic dcterms:hasPart ?scene .
  ?scene rdf:type parlipro:Scene .
  ?scene dcterms:hasPart ?speech .}}
GROUP BY ?topic ?member ?speech}
FILTER (?numOfFemales = 1 )}
GROUP BY ?topic ?member
```

F0: What members were born after 1950, their parties and dates of death if exist?

```
SELECT DISTINCT ?member ?party ?birthday
WHERE {
  ?speech rdf:type parlipro:Speech .
  ?speech parlipro:refMember ?member.
  ?speech parlipro:refParty ?party .
  ?member rdf:type parlipro:ParliamentMember .
  ?member bio:biography _:bio .
  _:bio rdf:type bio:Biography .
  _:bio foaf:birthday ?birthday .
FILTER (year(?birthday) > 1960)}
FILTER (year(?birthday) > 1950)
OPTIONAL{_:bio dbpedia-ont:deathDate ?deathDate .}}
```

F1: What is the gender of politicians who spoke most between 1995 and 2005?

```
SELECT ?gender COUNT(?member) AS ?numOfMembers
WHERE {
  ?proc rdf:type parlipro:ParliamentaryProceedings .
  ?proc dc:date ?date .
  ?proc dcterms:hasPart ?topic .
  ?speech rdf:type parlipro:Speech .
  ?speech parlipro:refMember ?member .
  ?member rdf:type parlipro:ParliamentMember .
  ?member bio:biography _:bio .
  _:bio rdf:type bio:Biography .
  _:bio foaf:gender ?gender .
  {?topic dcterms:hasPart ?speech .}
UNION{
  ?topic dcterms:hasPart ?sd .
  ?sd rdf:type parlipro:StageDirection .
  ?sd dcterms:hasPart ?speech .}
UNION{
  ?topic dcterms:hasPart ?scene .
  ?scene rdf:type parlipro:Scene .
  ?scene dcterms:hasPart ?speech .}
FILTER (year(?date) > 1995 AND year(?date) < 2005)}
GROUP BY ?gender
ORDER BY DESC(?numOfMembers)
LIMIT 1
```

F2: What is the percentage of male speakers?

```
SELECT (?numOfMaleMembers*100)/?numOfMembers
WHERE{{
SELECT COUNT(DISTINCT ?memberMale) as ?numOfMaleMembers
       COUNT(DISTINCT ?member) as ?numOfMembers
WHERE {{
   ?speech rdf:type parlipro:Speech .
   ?speech parlipro:refMember ?member .
   ?member rdf:type parlipro:ParliamentMember .}
UNION{
   ?memberMale bio:biography _:bio .
   _:bio rdf:type bio:Biography .
   _:bio foaf:gender dbpedia:Male .
FILTER (sameTerm(?member,?memberMale))}}}}
```

F3: What is the percentage of female speakers?

```
SELECT (?numOfFemaleMembers*100)/?numOfMembers
WHERE{{
SELECT COUNT(DISTINCT ?memberFemale) as ?numOfFemaleMembers
       COUNT(DISTINCT ?member) as ?numOfMembers
WHERE {{
  ?speech rdf:type parlipro:Speech .
  ?speech parlipro:refMember ?member .
  ?member rdf:type parlipro:ParliamentMember .}
UNION{
  ?memberFemale bio:biography _:bio .
  _:bio rdf:type bio:Biography .
  _:bio foaf:gender dbpedia:Female .
FILTER (sameTerm(?member,?memberFemale))}}}}
```

F4: What politician has most number of Wikipedia pages in different languages?

```
SELECT ?member ?numOfPages
WHERE {{
SELECT DISTINCT ?member COUNT(?dbpediaMember) AS ?numOfPages
WHERE {
  ?member rdf:type parlipro:ParliamentMember .
  ?member owl:sameAs ?dbpediaMember .}
GROUP BY ?member}}
ORDER BY DESC(?numOfPages)
LIMIT 1
```

F5: What speeches are made by politicians without Wikipedia pages?

```
SELECT DISTINCT ?speech ?member
WHERE {
  ?speech rdf:type parlipro:Speech .
  ?speech parlipro:refMember ?member .
  ?member rdf:type parlipro:ParliamentMember .
OPTIONAL {?member owl:sameAs ?dbpediaMember .}
FILTER (!bound(?dbpediaMember))}
```

T0: Retrieve top 10 members with the most number of speeches.

```
SELECT ?member COUNT(?speech) as ?numOfSpeeches
WHERE {
  ?member rdf:type parlipro:ParliamentMember .
  ?speech parlipro:refMember ?member .}
GROUP BY ?member
ORDER BY DESC(?numOfSpeeches)
LIMIT 10
```

T1: Retrieve top 10 topics when most of the people spoke.

```
SELECT ?topic COUNT(?member) as ?numOfMembersSpokeInTopic
WHERE {
  ?topic rdf:type parlipro:Topic .
  ?topic dcterms:hasPart ?speech .
  ?speech rdf:type parlipro:Speech .
  ?speech parlipro:refMember ?member .}
GROUP BY ?topic
ORDER BY DESC(?numOfMembersSpokeInTopic)
LIMIT 10
```

T2: Retrieve top 10 topics with the most number of speeches.

```
SELECT ?topic
COUNT(?speech) as ?numOfSpeeches
WHERE {
  ?topic rdf:type parlipro:Topic .
  ?speech rdf:type parlipro:Speech .
  {?topic dcterms:hasPart ?speech .}
UNION{
  ?topic dcterms:hasPart ?sd .
  ?sd rdf:type parlipro:StageDirection .
  ?sd dcterms:hasPart ?speech .}
UNION{
  ?topic dcterms:hasPart ?scene .
  ?scene rdf:type parlipro:Scene .
  ?scene dcterms:hasPart ?speech .}}
GROUP BY ?topic
ORDER BY DESC(?numOfSpeeches)
LIMIT 10
```

T3: Retrieve top 10 days with the most number of topics.

```
SELECT ?date COUNT(?topic) as ?numOfTopics
WHERE {
  ?proc rdf:type parlipro:ParliamentaryProceedings .
  ?proc dcterms:hasPart ?topic .
  ?proc dc:date ?date .}
GROUP BY ?date
ORDER BY DESC(?numOfTopics)
LIMIT 10
```

T4: Retrieve top 10 longest topics (i.e., topics with the most number of paragraphs).

```
SELECT ?topic COUNT(?par) as ?numOfPars
WHERE {
  ?topic rdf:type parlipro:Topic .
  ?speech rdf:type parlipro:Speech .
  ?speech dcterms:hasPart ?par .
  ?par rdf:type parlipro:Paragraph .
  {?topic dcterms:hasPart ?speech .}
UNION{
  ?topic dcterms:hasPart ?sd .
  ?sd rdf:type parlipro:StageDirection .
  ?sd dcterms:hasPart ?speech .}
UNION{
  ?topic dcterms:hasPart ?scene .
  ?scene rdf:type parlipro:Scene .
  ?scene dcterms:hasPart ?speech .}}
GROUP BY ?topic
ORDER BY DESC(?numOfPars)
LIMIT 10
```

D Test Machine Specification

For the benchmark evaluation we used a personal laptop Apple MacBook Pro. The operating system running is Mac OS X Lion 10.7.5 x64. The specification of the machine is the following:

Hardware

- CPUs: 2.8 GHz Intel Core i7 (2x2 cores)
- Memory: 8 GB 1333 MHz DDR3
- Hard Disk: 750GB

Software

- OpenLink Virtuoso: Open Source Edition v.06.01.3127 compiled from source for OS X
- MySQL Community Server (GPL) v. 5.5.15
- Scripts (bash 3.2, Python 2.7.3) to scale and upload RDF data sets, to create permutations of queries and run them on Virtuoso. The scripts are available for downloading at `http://data.politicalmashup.nl/RDF/scripts/`.

Virtuoso Configuration We configured the Virtuoso Server to handle load of large data sets[18].

[18] The configuration was performed following the guidelines at `http://www.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtRDFPerformanceTuning`

```
NumberOfBuffers = 680000
MaxDirtyBuffers  = 500000
```

The RDF Index Scheme remained as it was supplied with the default Virtuoso installation. Namely, the scheme consists of the following indexes:

- `PSOG` - primary key.
- `POGS` - bitmap index for lookups on object value.
- `SP` - partial index for cases where only S is specified.
- `OP` - partial index for cases where only O is specified.
- `GS` - partial index for cases where only G is specified.