Distributed reasoning: because size matters

Andreas Harth, Aidan Hogan, Spyros Kotoulas, Jacopo Urbani

Outline

- (This morning) Introduction to Linked Data
  - Foundations and Architectures
  - Crawling and Indexing
  - Querying
- (This morning) Integrating Web Data with Reasoning
  - Introduction to RDFS/OWL on the Web
  - Introduction and Motivation for Reasoning
- (Now!) Distributed Reasoning: Because Size Matters
  - Problems and Challenges
  - MapReduce and WebPIE
  - Demo
The Semantic Web growth

- Exponential growth of RDF
  - 2007: 0.5 Billion triples
  - 2008: 2 Billion triples
  - 2009: 6.7 Billion triples
  - 2010: 26.9 Billions triples
  - Now: ??

(Thanks to Chris Bizet for providing these numbers)
As of March 2009
PROBLEMS AND CHALLENGES
Problems and challenges

- One machine is not enough to store and process the Web
- We must distribute **data** and **computation**
- What architecture?
  - Several architectures of supercomputers
    - SIMD (single instruction/multiple data) processors, like graphic cards
    - Multiprocessing computers (many CPU shared memory)
    - Clusters (shared nothing architecture)
  - Algorithms depend on the architecture
- Clusters are becoming the reference architecture for High Performance Computing
Problems and challenges

- In a distributed environment the increase of performance comes at the price of new problems that we must face:
  - Load balancing
  - High I/O cost
  - Programming complexity
Problems and challenges: load balancing

- **Cause:** In many cases (like reasoning) some data is needed much more than other (e.g. schema triples)
- **Effect:** some nodes must work more to serve the others. This hurts scalability
Problems and challenges: high I/O cost

- **Cause:** data is distributed on several nodes and during reasoning the peers need to heavily exchange it
- **Effect:** hard drive or network speed become the performance bottleneck
Problems and challenges: programming complexity

- **Cause:** in a parallel setting there are many technical issues to handle
  - Fault tolerance
  - Data communication
  - Execution control
  - Etc.

- **Effect:** Programmers need to write much more code in order to execute an application on a distributed architecture
CURRENT STATE OF THE ART
What is the current State of The Art?

- From the beginning, some of the RDF Data store vendors support reasoning:
    - Support backward reasoning using OWL logic
  - 4Store [http://4store.org/](http://4store.org/)
    - Perform backward RDFS reasoning, works on clusters, up to 32 nodes
    - Support reasoning up to OWL 2 R, works on a single machine
    - Performs RDFS+ and custom rules
What is the current State of The Art?

- Also the research community has extensively worked on this problem
  - **MaRVIN (ASWC 2008)**: p2p network, RDFS reasoning
  - **Reasoning on cluster/Blue Gene (ISWC 2009)**: supercomputer, RDFS reasoning
  - **WebPIE (ESWC 2010 - JWS (u.p))**: MapReduce reasoner that supports OWL reasoning
  - **QueryPIE (@ ISWC 2011)**: works on a cluster, OWL ter Horst support, backward-chaining

- There is an European project specifically targeted to the problem of large-scale reasoning: **LarKC (topic of the next talk)**
What is the current State of The Art?

What performs the best?
- Web-scale reasoning means forward-chaining
- Currently WebPIE has shown the best scalability (100B triples)
- Alternative backward-chaining techniques like QueryPIE might change this trend in the future…

The topic of the remaining of this talk will be WebPIE
- 1) explain MapReduce
- 2) Show how WebPIE works
- 3) ive an idea of the performance
- 4) show how you can run it yourself
MAPREDUCE
MapReduce

- Analytical tasks over very large data (logs, web) are always the same
  - Iterate over large number of records
  - Extract something interesting from each
  - Shuffle and sort intermediate results
  - Aggregate intermediate results
  - Generate final output

**Idea: provide functional abstraction of these two functions**
MapReduce

- In 2004 Google introduced the idea of MapReduce
  - Computation is expressed only with Maps and Reduce
  - Hadoop is a very popular open source MapReduce implementation

- A MapReduce framework provides
  - Automatic parallelization and distribution
  - Fault tolerance
  - I/O scheduling
  - Monitoring and status updates

- Users write MapReduce programs -> framework executes them

http://hadoop.apache.org/
MapReduce

- A MapReduce program is a sequence of one (or more) map and a reduce function
- All the information is expressed as a set of key/value pairs
- The execution of a MapReduce program is the following:

  1. **map** function transforms input records in intermediate key/value pairs
     ```
     map(null, file) {
       for (word in file) output(word, 1)
     }
     ```
  2. MapReduce framework automatically groups the pairs
     ```
     reduce(word, set<numbers>) {
       int count = 0;
       for (int value : numbers) count += value;
       output(word, count)
     }
     ```
  3. **reduce** function processes each group and returns output
- Example: suppose we want to calculate the occurrences of words in a set of documents.
MapReduce

“How can MapReduce help us solving the three problems of above?”

- **High communication cost**
  - The map functions are executed on local data. This reduces the volume of data that nodes need to exchange

- **Programming complexity**
  - In MapReduce the user needs to write only the map and reduce functions. The frameworks takes care of everything else.

- **Load balancing**
  - This problem is still not solved. 😞 Further research is necessary…
WEBPIE
WebPIE

WebPIE is a forward reasoner that uses MapReduce to execute the reasoning rules.

All code, documentation, tutorial etc. is available online.

WebPIE algorithm:

- **Input**: triples in N-Triples format
  - 1) Compress the data with dictionary encoding
  - 2) Launch reasoning
  - 3) Decompress derived triples

- **Output**: triples in N-Triples format

http://cs.vu.nl/webpie/
WebPIE 1st step: compression

- Compressing the data is necessary to improve the performance
- In WebPIE we compress the data using dictionary encoding
- Why dictionary encoding and not simply zip?
  - Data becomes application inaccessible
  - With dictionary encoding apps can still manage data
- Why MapReduce for compression?
  - Data is too large for one machines
  - Dictionary table is too large to fit in memory
- Dictionary encoding with MapReduce is challenging!
  - Load balancing due to high data skew
  - Centralized dictionary encoding is a bottleneck in a distribute system
WebPIE: compression

- In WebPIE we solved the load balancing problem processing popular terms in the map and others in the reduce.
- Also, the centralized dictionary is replaced by partitioning numbers and assigning in parallel.
- Ok, but how does it work?
WebPIE 1\textsuperscript{st} step: compression

- **Compression algorithm**: sequence of 3 MapReduce jobs
  - 1\textsuperscript{st} job: identify popular term and assign a number
  - 2\textsuperscript{nd} job: deconstruct statement: replace popular terms in the map and not popular terms in the reduce
  - 3\textsuperscript{rd} job: reconstruct statement in compressed format

- **Decompression algorithm**: sequence of 4 MapReduce jobs
  - 1\textsuperscript{st} job: identify popular terms
  - 2\textsuperscript{nd} job: joins popular terms with dictionary
  - 3\textsuperscript{rd} job: deconstruct statements and replace popular terms in map and not popular in reduce
  - 4\textsuperscript{th} job: reconstruct statement in N-Triples format
WebPIE 2nd step: reasoning

- Reasoning means applying a set of rules on the entire input until no new derivation is possible.
- The difficulty of reasoning depends on the logic considered.
- RDFS reasoning
  - Set of 13 rules
  - All rules require at most one join between a “schema” triple and an “instance” triple.
- OWL reasoning
  - Logic more complex => rules more difficult
  - The ter Horst fragment provides a set of 23 new rules
  - Some rules require a join between instance triples
  - Some rules require multiple joins
WebPIE 2\textsuperscript{nd} step: reasoning

- Reasoning means applying a set of rules on the entire input until no new derivation is possible
- The difficulty of reasoning depends on the logic considered
- **RDFS reasoning**
  - Set of 13 rules
  - All rules require at most one join between a “schema” triple and an “instance” triple
- **OWL reasoning**
  - Logic more complex => rules more difficult
  - The ter Horst fragment provides a set of 23 new rules
  - Some rules require a join between instance triples
  - Some rules require multiple joins
WebPIE 2nd step: RDFS reasoning

Q: How can we apply a reasoning rule with MapReduce?
A: During the map we write in the intermediate key matching point of the rule and in the reduce we derive the new triples

Example: if a rdf:type B
and B rdfs:subClassOf C
then a rdf:type C
WebPIE 2nd step: RDFS reasoning

- However, such straightforward way does not work because of several reasons
  - Load balancing
  - Duplicates derivation
  - Etc.

- In WebPIE we applied three main optimizations to apply the RDFS rules
  1. We apply the rules in a specific order to avoid loops
  2. We execute the joins replicating and loading the schema triples in memory
  3. We perform the joins in the reduce function and use the map function to generate less duplicates
WebPIE 2\textsuperscript{nd} step: RDFS reasoning

1\textsuperscript{st} optimization: apply rules in a specific order
WebPIE 2\textsuperscript{nd} step: RDFS reasoning

- 2\textsuperscript{nd} optimization: perform the join during the map
  - The schema is small enough to fit in memory
  - Each node loads them in memory
  - The instance triples are read as MapReduce input and the join is done against the in-memory set

- 3\textsuperscript{rd} optimization: avoid duplicates with special grouping
  - The join can be performed either in the map or in the reduce
  - If we do it in the reduce, then we can group the triples so that the key is equal to the derivation part that is input dependent.
  - Groups cannot generate same derived triple => no duplicates
WebPIE 2\textsuperscript{nd} step: reasoning

- Reasoning means applying a set of rules on the entire input until no new derivation is possible.
- The difficulty of reasoning depends on the logic considered.
- RDFS reasoning
  - Set of 13 rules.
  - All rules require at most one join between a “schema” triple and an “instance” triple.
- OWL reasoning
  - Logic more complex => rules more difficult.
  - The ter Horst fragment provides a set of 23 new rules.
  - Some rules require a join between instance triples.
  - Some rules require multiple joins.
WebPIE 2\textsuperscript{nd} step: OWL reasoning

- Since the RDFS optimizations are not enough, we introduced new optimizations to deal with the more complex rules.
- We will not explain all of them, but only one.
- Example: if \( \text{<p type TransitiveProperty>} \) and \( \text{<a p b>} \) and \( \text{<b p c>} \), then \( \text{<a p c>} \)

This rule is problematic because
- Need to perform join between instance triples
- Every time we derive also what we derived before
- Solution: we perform the join in the “naïve” way, but we only consider triples on a “specific” position.
WebPIE 2nd step: OWL reasoning

Example

**Input:**

\{ <a \: p \: b>, 1 \}
\{ <b \: p \: c>, 1 \}
\{ <c \: p \: d>, 1 \}
\{ <d \: p \: e>, 1 \}
\{ <e \: p \: f>, 1 \}

1st M/R job

**Output:**

\{ <a \: p \: c>, 2 \}
\{ <b \: p \: d>, 2 \}
\{ <c \: p \: e>, 2 \}
\{ <d \: p \: f>, 2 \}
WebPIE 2\textsuperscript{nd} step: OWL reasoning

Example

After job 1:

\{ <b \text{p} c>, 1 \} \\
\{ <a \text{p} b>, 1 \} \\
\{ <b \text{p} d>, 2 \} \\
\{ <d \text{p} f>, 2 \} \\
... \\

2nd M/R job

Output:

\{ <b \text{p} d>, 3 \} \\
\{ <b \text{p} f>, 4 \}
WebPIE 2nd step: OWL reasoning

- By accepting only triples with a specific distance we avoid to derive information already derived
- General rule:
  - Every job accepts in input only triples derived in the previous two steps
  - During the execution of the \( nth \) job we derive only if:
    - The antecedent triples on the left side have distance \( 2^{(n-1)} \) or \( 2^{(n-2)} \)
    - The antecedent triples on the right side have distance greater than \( 2^{(n-2)} \)
WebPIE: performance

- We tested the performance on LUBM, LDSR, Uniprot
- Tests were conducted at the DAS-3 cluster (http://www.cs.vu.nl/das)
- Performance depends not only on input size but also the complexity of the input
- Execution time using 32 nodes:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Input</th>
<th>Output</th>
<th>Exec. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUBM</td>
<td>1 Billion</td>
<td>0.5 Billion</td>
<td>1 Hour</td>
</tr>
<tr>
<td>LDSR</td>
<td>0.9 Billion</td>
<td>0.9 Billion</td>
<td>3.5 Hours</td>
</tr>
<tr>
<td>Uniprot</td>
<td>1.5 Billion</td>
<td>2 Billions</td>
<td>6 Hours</td>
</tr>
</tbody>
</table>
WebPIE: performance

- Input complexity

Reasoning on LUBM

Reasoning on LDSR
WebPIE: performance

- Scalability (on the input size, using LUBM to 100 Billion triples)
WebPIE: performance

![Graph showing throughput (K.Triples/Sec) vs. input size (Billions of statements) for different systems: BigOWLIM, Oracle 11g, DAML DB, BigData. The graph illustrates performance comparison between these systems as input size increases.]
WebPIE: performance (in 2010)

We are here!!
WebPIE: performance (in 2011)

Now we are here!

- BigOWLIM
- Oracle 11g
- DAML DB
- BigData
- WebPIE (DAS4)

2010
Conclusions

- With WebPIE we show that high performance reasoning on very large data is possible
- We need to compromise w.r.t. reasoning complexity and performance
- Still many problems unresolved:
  - How do we collect the data?
  - How do we query large data?
  - How do we introduce a form of authoritative reasoning to prevent “bad” derivation?
  - Etc.
QUESTIONS?