A Linked Data Reasoner in the Cloud

Jules Chevalier
jules.chevalier@univ-st-etienne.fr

LT2C, Télécom Saint Etienne, Université Jean Monnet

December 2014

Supervisors :
Frédérique Laforest
Christophe Gravier
Julien Subercaze

Education

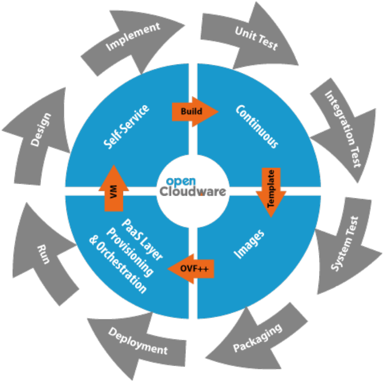
- 2010-2012 : Master Degree "Web Intelligence", UJM
- Feb-Jul 2012 : Research Internship "Knowledge in the Cloud", LT2C
- 2012-2015 : PhD Thesis "Distributed Reasoning", LT2C

Research Intership

- Skill improvement in Semantic Web and Inference Process
- State of the art
- Outline of proposition
- Implementation tests

Supervisors : Christophe Gravier
Julien Subercaze

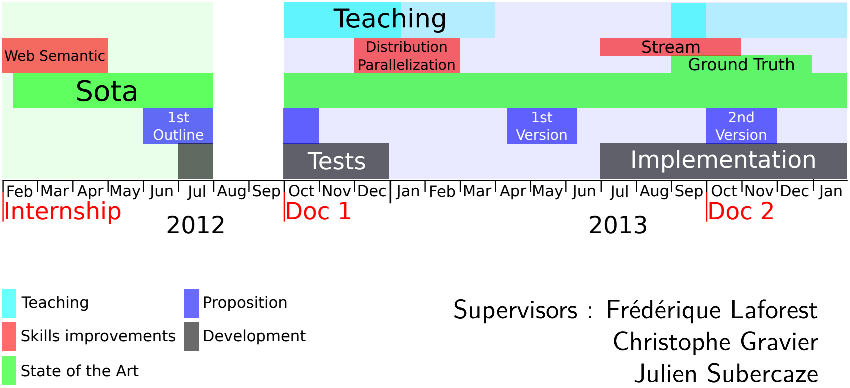
OpenCloudware aims at building an open software engineering platform, for the collaborative development of distributed applications to be deployed on multiple Cloud infrastructures.



source: *http://www.opencloudware.org*

Thesis : Distributed Reasoning

- From 1st October 2012 to October 2015
- Continuation of the state of the art
- Improving the proposition
- Beginning of the implementation



Supervisors : Frédérique Laforest
Christophe Gravier
Julien Subercaze

Summary

Semantic Web & Description Logic

- Formalises concepts to represent them
- Standardizes this representation
- Makes it readable for both humans and computers
- Link these data together
- Allows automatic operations on these data
  - Integrity constraint validation
  - Explicit implicit data from the base
  - Query the knowledge base

Semantic Web & Description Logic

- Formalises concepts to represent them
- Standardizes this representation
- Makes it readable for both humans and computers
- Link these data together
- Allows automatic operations on these data
  - Integrity constraint validation
  - **Explicit implicit data from the base**
  - Query the knowledge base

Problematic

Problematic

- Reasoning process scaling to Big Data (NP complete for most complex ontologies)

Problematic

Problematic

- Reasoning process scaling to Big Data (NP complete for most complex ontologies)

Idea

- Distribute the inference process among several nodes
- Use the Cloud as runtime environment
  - Flexibility: Adapt the number/power of nodes to the needs
  - Cost limitation: We pay what we use
  - Low latency between nodes in the Cloud

Summary

Logic Description [6]

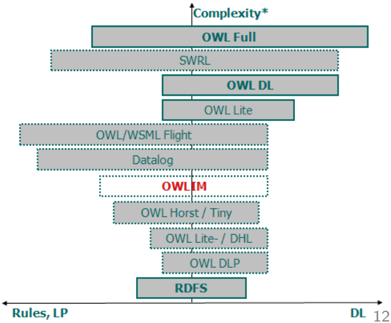
- Representation of data
  - Understandable by both humans and machines
  - Formal
  - Universal
- Interpretations
  - **Open World Assumption** : Everything exists until something says it's not
  - **Close World Assumption** : The world is limited by the definitions

Fragments

- A fragment is a list of axioms
- Semantic Web standards suggest different pre defined fragments (RDFS, OWL Lite, OWL Full, OWL DL, ...)
- The more they have a high expressivity, the more the operations are complex (from P to NEXPTIME)
- Choosing one fragment is trade off between expressivity and computational complexity

Example:

- `domain`
- `range`
- `subClassOf`
- `subPropertyOf`
- `type`



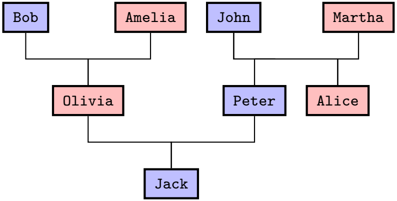
Ontology example : TBox and TBox

TBox : Definitions

$\text{Man} \equiv \text{Human} \sqcap \text{Male}$	*A Man is a Male Human*
$\text{Woman} \equiv \text{Human} \sqcap \neg\text{Male}$	*A Woman is a non Male Human*
$\text{Parent} \equiv \exists\text{hasChild}.\top$	*A parent has at least one Child*
$\text{Father} \equiv \text{Parent} \sqcap \text{Man}$	*A Father is a Man Parent*
$\text{Mother} \equiv \text{Parent} \sqcap \text{Woman}$	*A Mother is a Woman Parent*

ABox : Individuals

Man	`Bob, John, Jack, Peter, Alfred`
Woman	`Olivia, Astrid, Amelia, Alice, Martha`
hasChild	`(Bob,Olivia), (Amelia,Olivia), (John,Alice), (John,Peter), (Martha,Alice), (Martha,Guillaune), (Olivia,Jack), (Peter,Jack)`



Knowledge processing

- The logic description allows several operations on the Knowledge Base [5, 1] :
  - Consistency checking
  - Satisfiability checking
  - Querying
  - Classification
  - **Reasoning/Inference**
  - ...

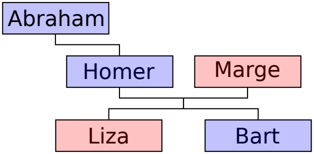
Inference rules

ρdf [7] rules (`type`, `subClassOf`, `subPropertyOf`, `domain`, `range`)

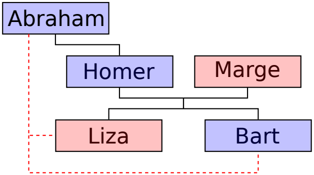
CAX-SCO	c_1 rdfs:subClassOf c_2 x rdf:type c_1	x rdf:type c_2
PRP-DOM	p rdfs:domain c x p y	x rdf:type c
PRP-RNG	p rdfs:range c x p y	y rdf:type c
SCM-SCO	c_1 rdfs:subClassOf c_2 c_2 rdfs:subClassOf c_3	c_1 rdfs:subClassOf c_3
SCM-EQC2	c_1 rdfs:subClassOf c_2 c_2 rdfs:subClassOf c_1	c_1 owl:equivalentClass c_2
SCM-DOM1	p rdfs:domain c_1 c_1 rdfs:subClassOf c_2	p rdfs:domain c_2
SCM-RNG1	p rdfs:range c_1 c_1 rdfs:subClassOf c_2	p rdfs:range c_2
PRP-SPO1	p_1 rdfs:subPropertyOf p_2 x p_1 y	x p_2 y
SCM-SPO	p_1 rdfs:subPropertyOf p_2 p_2 rdfs:subPropertyOf p_3	p_1 rdfs:subPropertyOf p_3
SCM-DOM2	p_2 rdfs:domain c p_1 rdfs:subPropertyOf $p2$	p_1 rdfs:domain c
SCM-RNG2	p_2 rdfs:range c p_1 rdfs:subPropertyOf p_2	p_1 rdfs:range c
SCM-EQP2	p_1 rdfs:subPropertyOf p_2 p_2 rdfs:subPropertyOf p_1	p_1 owl:equivalentProperty p_2

Reasoning : Forward Chaining VS Backward Chaining

- What we know :
  - Abraham father Homer
  - Homer father Liza
  - Homer father Bart
  - Marge mother Liza
  - Marge mother bart

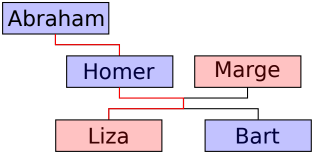


Reasoning : Forward Chaining VS Backward Chaining



- What we know :
  - Abraham father Homer
  - Homer father Liza
  - Homer father Bart
  - Marge mother Liza
  - Marge mother bart
- What **Forward Chaining** do :
  - Abraham grandfather Liza
  - Abraham grandfather Bart
  - ...
  - Abraham grandfather Liza ? $\rightarrow$ yes

Reasoning : Forward Chaining VS Backward Chaining



- What we know :
  - Abraham father Homer
  - Homer father Liza
  - Homer father Bart
  - Marge mother Liza
  - Marge mother bart
- What **Forward Chaining** do :
  - Abraham grandfather Liza
  - Abraham grandfather Bart
  - ...
  - Abraham grandfather Liza ? $\rightarrow$ yes
- What **Backward Chaining** do :
  - Abraham grandfather Liza ?
  - Abraham father X & X father Liza ?
  - Abraham father Homer & Homer father Liza $\rightarrow$ yes

Our problematic

What we want to do

- Forward chaining for fast query answers
- Fragment agnostic
- Customizable rule-based inference
- Constraints bounded inference (time/number of triples)

Our problematic

What we want to do

- Forward chaining for fast query answers
- Fragment agnostic
- Customizable rule-based inference
- Constraints bounded inference (time/number of triples)

What are the problems

- Rules form a cyclic graph
  - Complexity depends on the fragment !
- The amount of triples generated is quite unpredictable
  - The complexity also depends on data !

Our problematic

What we want to do

- Forward chaining for fast query answers
- Fragment agnostic
- Customizable rule-based inference
- Constraints bounded inference (time/number of triples)

What are the problems

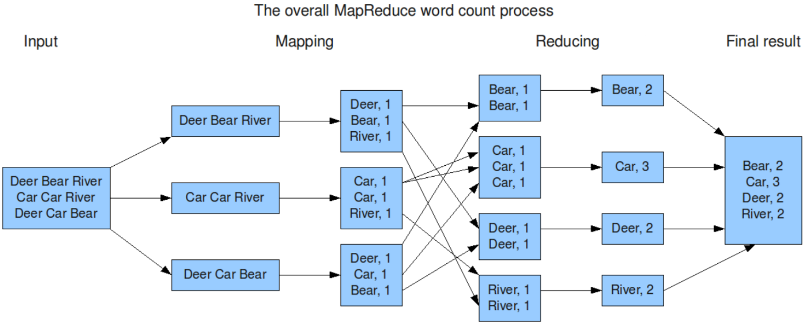
- Rules form a cyclic graph
  - Complexity depends on the fragment !
- The amount of triples generated is quite unpredictable
  - The complexity also depends on data !

Outlines to resolve them

- Distribute the process
- Optimise the rules schedule
- Help the user choosing wisely the fragment

Summary

Introduction - MapReduce [3]



source: *blog.jteam.nl/2009/08/04/introduction-to-hadoop*

MapReduce approaches

WebPie : a Web-scale Parallel Inference Engine

- 2009 - Jacopo Urbani Thesis [11]
  - Uses MapReduce for OWL Horst and RDFS reasoning
- 2011 - Fix some issues to improve OWL Horst reasoning [12]
  - Duplicates limitation
  - Indexation for `sameAs`
  - *Greedy* scheduling
  - Cleaner Job after some rules, or at the end

MapResolve [9]

- Based their work on WebPie to develop a OWL Horst reasoner
- Use 3 sets for triples : usable, used, inferred
- Limit the amount of data processed at the same time
- Points out MapReduce limitations

Parallel Inferencing for OWL Knowledge Bases [10]

- Proposes two ways to distribute the process

Split the data

- By graph partition
- By hash
- By domain expert knowledges

Split the rules

- By graph partition

- Gives a distributed algorithm using a external reasoner

Analysis : MapReduce approaches

MapReduce Framework

- Allows to implement distributed tasks
- The Hadoop framework
- Best suited to batch process huge amounts of data

- MapReduce requires an acyclic dataflow
- Jobs run in isolation
- Not suitable network shuffling
- Hadoop distributed file system

WebPie and MapResolve Contributions

- Despite optimisations, performances are low
- Nodes must wait for each other
- Generates a lot of duplicates
- Fragment dependant
- Naive partitioning

- Critical letter for WebPie [8]

Analysis : Parallel Inferencing for OWL Knowledge Bases

- Proposes smart partitioning
- Integrates an existing reasoner
- Fragment agnostic

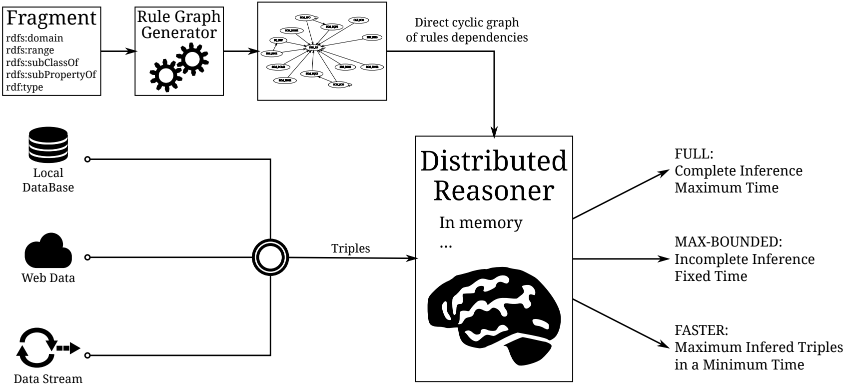
- Data are still in hermetic cores
- No shared data
- Nodes must wait to receive the new generated triples

Summary

Main lines

- Avoid data isolation with triples flow
- Data structures *for* distributed reasoning
- Fragment/source agnostic
- Smart adaptive scheduling
- Three reasoning options : `Full`, `Max-Bounded`, `Faster`

General Outline



Avoid batch process with flow management

- MapReduce is not well suited for inference because of data isolation
- We manage triples flow to :
  - Adapt the node network
  - Avoid data isolation
  - Prevent nodes awaiting
- Calls for stream processing of triples
  - Batch $\sqsubseteq$ Stream
  - But Stream $\not\sqsubseteq$ Batch ! (Especially `Max-Bounded` and `Faster` problems)

Avoid batch process with flow management

- MapReduce is not well suited for inference because of data isolation
- We manage triples flow to :
  - Adapt the node network
  - Avoid data isolation
  - Prevent nodes awaiting
- Calls for stream processing of triples
  - Batch $\sqsubseteq$ Stream
  - But Stream $\not\sqsubseteq$ Batch ! (Especially `Max-Bounded` and `Faster` problems)

As of today

- Reasoner input can be batched or streamed
- Rules are flow based
- Multiple instances of rules can run at the same time

Data structures for distributed reasoning

Numbers Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

source: *Software Engineering Advice from Building Large-Scale Distributed Systems*, Jeff Dean

Data structures for distributed reasoning

Numbers Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	**10,000,000 ns**
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

source: *Software Engineering Advice from Building Large-Scale Distributed Systems*, Jeff Dean

Data structures for distributed reasoning

Semantic Web is not Big Data

- RAM faster than disk
- HDFS was conceived for PetaBytes processing
- Does large datasets fit in RAM ?

Data structures for distributed reasoning

Semantic Web is not Big Data

- RAM faster than disk
- HDFS was conceived for PetaBytes processing
- Does large datasets fit in RAM ?
- Billion Triples Challenge 2012 Dataset : 1.4 Billion triples
- 3 long per triples : 192 bits

Data structures for distributed reasoning

Semantic Web is not Big Data

- RAM faster than disk
- HDFS was conceived for PetaBytes processing
- Does large datasets fit in RAM ?
- Billion Triples Challenge 2012 Dataset : 1.4 Billion triples
- 3 long per triples : 192 bits
- 1.4 Billion triples fits in 33GB of RAM

Data structures for distributed reasoning

Semantic Web is not Big Data

- RAM faster than disk
- HDFS was conceived for PetaBytes processing
- Does large datasets fit in RAM ?
- Billion Triples Challenge 2012 Dataset : 1.4 Billion triples
- 3 long per triples : 192 bits
- 1.4 Billion triples fits in 33GB of RAM
- After inference : 27 Billion triples
- Fits in 1TB (cost $\leq$ 10,000$)

Data structures for distributed reasoning

- We need efficient distributed structures with :
  - Concurrent structures
  - Indexed structures for fast retrieving
  - In memory storage for faster access
- With support for :
  - Network exchange
  - Shared data
  - Minimum disk access (loading only)

As of today

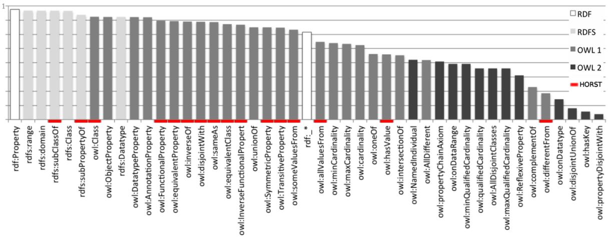
- TripleStore object shared by threads
  - Concurrent
  - Indexed
  - Immutable triples
- Every running rule can access it
- Direct access limits duplicates

Fragment agnostic

- Custom fragment or rule set
- The user can also define new custom rules
- Individual rules optimisations
- Proposes fragment optimisations

As of today

- Any rule is a distinct class
- Dynamic schedule
- Can easily add new rules



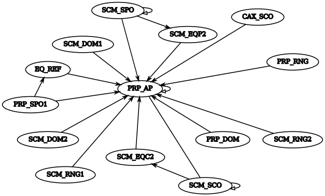
Rank of most used concepts usable in inference, with fragments highlighting (data from [4])

Smart adaptive scheduling

- Automatically schedules the rules thanks to the dependence rules graph
- The schedule depends on the fragment and so on the set of rules
- Just-in-time scheduling

As of today

- Schedules the rules at the beginning
- Only take into account the specified rules

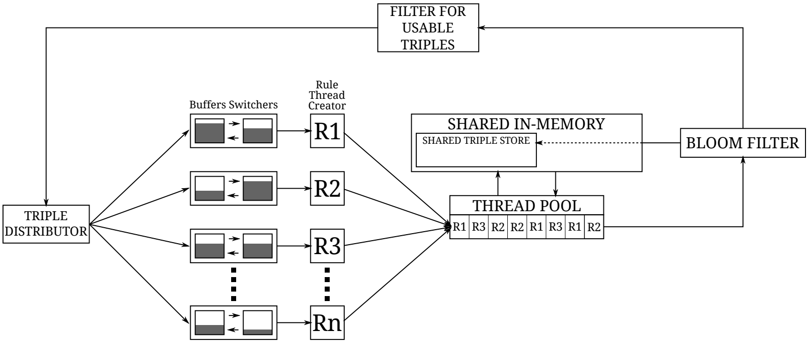


Multithreaded version

- Before implement distributed version, we start by a Multithreaded one, and then upgrade to distributed architecture.

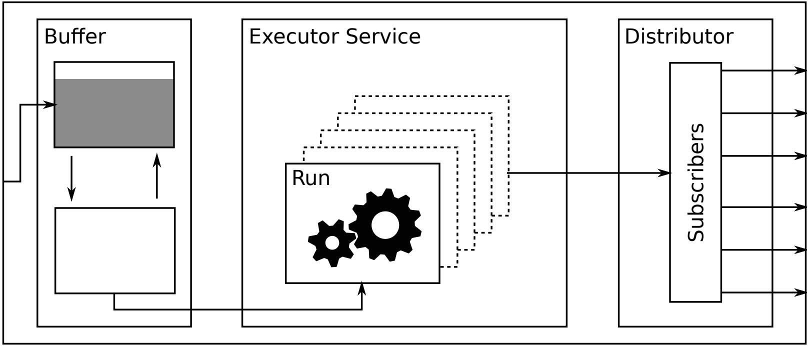
Outline evolution

First reasoner outline



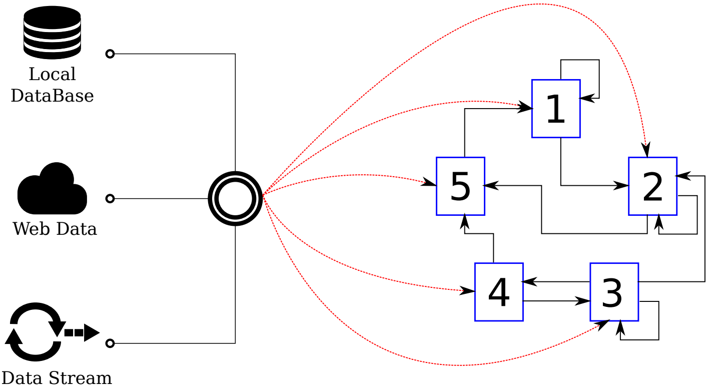
Outline evolution

Rule streamed evolution



Outline evolution

Actual reasoner outline



What's new

- There is not one node dedicated to one task
- New thread pre allocated, dormant
- Automatic rules scheduling
- Shared data in memory
- Streaming process

Summary

Publications

ESWC 2013

Accepted paper *A Linked Data Reasoner in the Cloud* [2]

Poster presentation at the PhD Symposium

RR 2013

Summer School participation

Poster presentation at the PhD Symposium

Schedule

- End of 2013 - First streamed version
  - WebPie deployment and tests - Our baseline
- February 2014 - Proposal of our Cloud-hosted linked data reasoner
  - Extended analysis of 1st version *vs* baseline
- May 2014 - First implementation with smart scheduling
  - Extended analysis of smart version *vs* 1st version
- November 2014 - Reasoner open-sourced with documentation
  - Tests launched for 3 addressed problems
- January 2015 - Additional features :
  - Help the user choosing a fragment
  - Propose fragment optimisations
  - Make predictions about the inference (time and space)
  - Make optimisation on the reasoner itself
- April 2015 - Writing the PhD thesis

Bibliography I

[1] Baader, F.
Tableau algorithms for description logics.
Automated Reasoning with Analytic Tableaux and (2000), 1–18.

[2] Chevalier, J.
A Linked Data Reasoner in the Cloud.
The Semantic Web: Semantics and Big Data 7882 (2013), 722–726.

[3] Dean, J., and Ghemawat, S.
MapReduce: Simplified data processing on large clusters.
Communications of the ACM 51, 1 (2008), 107–113.

[4] Glimm, B., Hogan, A., Krötzsch, M., and Polleres, A.
OWL: Yet to arrive on the Web of Data?
LDOW (2012).

[5] Krötzsch, M., Rudolph, S., and Hitzler, P.
Description logic rules.
2010.

[6] Mendelson, E.
Introduction to mathematical logic.
Syntax (2007).

[7] Muñoz, S., Pérez, J., and Gutierrez, C.
Minimal deductive systems for RDF.
The Semantic Web: Research and Applications 4519 (2007), 53–67.

[8] Patel-Schneider, P.
Letter: Comments on WebPIE: A Web-scale parallel inference engine using MapReduce.
Web Semantics: Science, Services and Agents on . . . 15 (Sept. 2012), 69–70.

Bibliography II

[9] Schlicht, A., and Stuckenschmidt, H.
Mapresolve.
Web Reasoning and Rule Systems 6902 (2011), 294–299.

[10] Soma, R., and Prasanna, V.
Parallel Inferencing for OWL Knowledge Bases.
2008 37th International Conference on Parallel Processing (Sept. 2008), 75–82.

[11] Urbani, J.
RDFS/OWL reasoning using the MapReduce framework.
PhD thesis, Vrije Universiteit - Faculty of Sciences, 2009.

[12] Urbani, J., Kotoulas, S., Maassen, J., van Harmelen, F., and Bal, H. E.
WebPIE: A Web-scale parallel inference engine using MapReduce.
J. Web Sem. 10 (2012), 59–75.