Towards Parallel Nonmonotonic Reasoning with Billions of Facts

Ilias Tachmazidis
Institute of Computer Science, FORTH
Department of Computer Science, University of Crete
tahmazid@ics.forth.gr

Grigoris Antoniou
Institute of Computer Science, FORTH
University of Huddersfield, UK
antoniou@ics.forth.gr

Giorgos Flouris
Institute of Computer Science, FORTH
fgeo@ics.forth.gr

Spyros Kotoulas
IBM Research, Ireland
Spyros.Kotoulas@ie.ibm.com

Abstract

We are witnessing an explosion of available data from the Web, government authorities, scientific databases, sensors and more. Such datasets could benefit from the introduction of rule sets encoding commonly accepted rules or facts, application- or domain-specific rules, commonsense knowledge etc. This raises the question of whether, how, and to what extent knowledge representation methods are capable of handling the vast amounts of data for these applications. In this paper, we consider nonmonotonic reasoning, which has traditionally focused on rich knowledge structures. In particular, we consider defeasible logic, and analyze how parallelization, using the MapReduce framework, can be used to reason with defeasible rules over huge data sets. Our experimental results demonstrate that defeasible reasoning with billions of data is performant, and has the potential to scale to trillions of facts.

1 Introduction

Recently, we experience an unprecedented increase in the quantity of available data consisting of raw data coming from sensor readings, data stored from scientific databases, governmental data etc. In most cases, such data are published on the Web, in order to facilitate exchange and interlinkage of knowledge ((Roussakis, Flouris, and Christophides 2011)). The recent rising of the Linked Open Data initiative[1] ((Bizer, Heath, and Berners-Lee 2009)) is an answer to the need for such large and interconnected data.

Traditionally, the area of knowledge representation has focused on complex knowledge structures and reasoning methods for processing these structures. The new arising challenge is to study how reasoning can process such interesting knowledge structures in conjunction with huge data sets. To fully exploit the immense value of such datasets and their interconnections, one should be able to reason over them using rule sets that would allow the aggregation, visualization, understanding and exploitation of the raw data that comprise the databases. Such reasoning is based on rules which capture the inference semantics of the underlying knowledge representation formalism, but also rules which encode commonsense, practical knowledge that humans possess and would allow the system to automatically reach useful conclusions based on the provided data and infer new and useful knowledge based on the data. For example, in ((Urbani et al. 2009)) for 78,8 million statements crawled from the Web, the number of inferred conclusions (RDFS closure) consists of 1,5 billion triples.

In this paper, we consider nonmonotonic rule sets ((Antoniou and van Harmelen 2008), (Maluszynski and Szalas 2010)). Such rule sets provide additional benefits because they are more suitable for encoding commonsense knowledge and reasoning. In addition, nonmonotonic rules avoid triviality of inference, which could easily occur when low-quality raw data is fed to the system; the latter is common in this setting, given the interconnection of data from different sources, over which the data engineer has no control.

The main challenge rising in such a setting is the feasibility of reasoning over such large volumes of data. One of the most promising methods to address this problem is by using massively parallel reasoning processes that would handle reasoning by using several computers in the cloud, assigning each of them a part of the parallel computation.

In the last two of years, there has been significant progress in parallel reasoning e.g., in ((Oren et al. 2009)), ((Urbani et al. 2009)), ((Kotoulas, Oren, and van Harmelen 2010)), ((Goodman et al. 2011))), scaling reasoning up to 100 billion triples ((Urbani et al. 2010)). Nevertheless, current approaches have been restricted to monotonic reasoning, namely RDFS and OWL-horst, or have not been evaluated for scalability ((Mutharaju, Maier, and Hitzler 2010)).

However, in many application scenarios, one needs to deal with poor quality data (e.g., involving inconsistency or incompleteness), which could easily lead to reasoning triviality when considering rules based on monotonic formalisms; this problem can be managed with nonmonotonic rules and nonmonotonic reasoning.

We study the problem of reasoning over huge datasets equipped with nonmonotonic (defeasible) rules using massively parallel (cloud) computational techniques. Following previous works, we adopt the MapReduce framework ((Dean and Ghemawat 2004)), suited for parallel processing of huge datasets.

A restricted form of defeasible logic is studied: single-argument defeasible logic. A MapReduce algorithm is pre-

[1]http://linkeddata.org/

sented, followed by an extensive experimental evaluation. The findings show that reasoning with billions of facts is possible for a variety of knowledge theories. Due to space restrictions the presentation of the work on multi-argument defeasible logic is deferred to a future paper.

To our best knowledge, this is the first work addressing nonmonotonic reasoning using mass parallelization techniques. As a long-term plan, we intend to apply mass parallelization to a variety of inconsistency handling logics. This would allow the use of defeasible reasoning as a nonmonotonic rule "layer" on the semantic web. Moreover, we plan to apply similar approaches to systems of argumentation, well-founded semantics, answer-set semantics ((Gelfond 2008)), and ontology ((Konstantinidis et al. 2008)) and repair ((Roussakis, Flouris, and Christophides 2011)).

The paper is organized as follows: Section 2 introduces the MapReduce Framework and Defeasible Logics. An algorithm for single-argument defeasible logic and its evaluation are presented in Sections 3 and 4 respectively.

2 Preliminaries

2.1 MapReduce Framework

MapReduce is a framework for parallel processing over huge data sets ((Dean and Ghemawat 2004)). Processing is carried out in a map and a reduce phase. For each phase, a set of user-defined map and reduce functions are run in parallel. The former performs a user-defined operation over an arbitrary part of the input and partitions the data, while the latter performs a user-defined operation on each partition.

MapReduce is designed to operate over key/value pairs. Specifically, each Map function receives a key/value pair and emits a set of key/value pairs. Subsequently, all key/value pairs produced during the map phase are grouped by their key and passed to reduce phase. During the reduce phase, a $Reduce$ function is called for each unique key, processing the corresponding set of values.

Let us illustrate the *wordcount* example. In this example, we take as input a large number of documents and calculate the frequency of each word. The pseudo-code for the Map and $Reduce$ functions is depicted in Algorithm 1.

Algorithm 1 Wordcount example

```
map(Long key, String value) :
   // key: position in document
   // value: document line
   for each word w in value
      EmitIntermediate(w, "1");

reduce(String key, Iterator values) :
   // key: a word
   // values : list of counts
   int count = 0;
   for each v in values
      count += ParseInt(v);
   Emit(key , count);
```

During map phase, each map operation gets as input a line of a document. Map function extracts words from each line and emits that word *w* occurred once (<w, 1>). Consider the line : "Hello... Hello!". Instead of emitting <Hello, 2>, Map function emits <Hello, 1> twice. As mentioned above, the MapReduce framework will group and sort pairs by their key. Specifically for the word *Hello*, a pair <Hello, <1,1>> will be passed to the $Reduce$ function. The $Reduce$ function has to sum up all occurrence values for each word emitting a pair containing the word and the frequency of the word. The final result for the word *Hello* will be <Hello, 2>.

2.2 Defeasible Logic

A defeasible theory D is a triple (F,R,>) where F is a finite set of facts (literals), R a finite set of rules, and > a superiority relation (acyclic relation upon R).

A rule r consists (a) of its antecedent (or body) A(r) which is a finite set of literals, (b) an arrow, and, (c) its consequent (or head) C(r) which is a literal. There are three types of rules: strict rules, defeasible rules and defeaters represented by a respective arrow $\rightarrow$, $\Rightarrow$ and $\rightsquigarrow$.

Given a set R of rules, we denote the set of all strict rules in R by R_s, and the set of strict and defeasible rules in R by R_{sd}. R[q] denotes the set of rules in R with consequent q. If q is a literal, $\sim$q denotes the complementary literal (if q is a positive literal p then $\sim$q is $\neg$p; and if q is $\neg$p, then $\sim$q is p).

A conclusion of D is a tagged literal and can have one of the following four forms:

- $+\Delta$q, which is intended to mean that q is definitely provable in D.
- $-\Delta$q, which is intended to mean that we have proved that q is not definitely provable in D.
- $+\partial$q, which is intended to mean that q is defeasibly provable in D.
- $-\partial$q, which is intended to mean that we have proved that q is not defeasibly provable in D.

Provability is defined below. It is based on the concept of a derivation (or proof) in D = (F, R, >). A derivation is a finite sequence P = P(1), ..., P(n) of tagged literals satisfying the following conditions. The conditions are essentially inference rules phrased as conditions on proofs. P(1..$\imath$) denotes the initial part of the sequence P of length i. More details on provability are omitted due to lack of space, but can be found in ((Maher 2004)).

$+\Delta$: We may append P($\imath$ + 1) = $+\Delta$q if either
  q $\in$ F or
  $\exists$r $\in$ R_s[q] $\forall\alpha \in$ A(r): $+\Delta\alpha \in$ P(1..$\imath$)

$-\Delta$: We may append P($\imath$ + 1) = $-\Delta$q if
  q $\notin$ F and
  $\forall$r $\in$ R_s[q] $\exists\alpha \in$ A(r): $-\Delta\alpha \in$ P(1..$\imath$)

$+\partial$: We may append P ($\imath$ + 1) = $+\partial$q if either
  (1) $+\Delta$q $\in$ P(1..$\imath$) or
  (2) (2.1) $\exists$r $\in$ R_{sd}[q] $\forall\alpha \in$ A(r): $+\partial\alpha \in$ P(1..$\imath$) and
    (2.2) $-\Delta$ $\sim$q $\in$ P(1..$\imath$) and
    (2.3) $\forall$s $\in$ R[$\sim$q] either
      (2.3.1) $\exists\alpha \in$ A(s): $-\partial\alpha \in$ P(1..$\imath$) or

(2.3.2) $\exists$t $\in$ R$_{sd}$[q] such that
$\forall\alpha \in$ A(t): $+\partial\alpha \in$ P(1..$\imath$) and t $>$ s

$-\partial$: We may append P($\imath$ + 1) = $-\partial$q if
(1) $-\Delta$q $\in$ P(1..$\imath$) and
(2) (2.1) $\forall$r $\in$ R$_{sd}$[q] $\exists\alpha \in$ A(r): $-\partial\alpha \in$ P(1..$\imath$) or
(2.2) $+\Delta$ $\sim$q $\in$ P(1..$\imath$) or
(2.3) $\exists$s $\in$ R[$\sim$q] such that
(2.3.1) $\forall\alpha \in$ A(s): $+\partial\alpha \in$ P(1..$\imath$) and
(2.3.2) $\forall$t $\in$ R$_{sd}$[q] either
$\exists\alpha \in$ A(t): $-\partial\alpha \in$ P(1..$\imath$) or t $\not>$ s

3 Algorithm description

The implementation for single-argument predicates is based on the combination of defeasible logic with MapReduce. In order to achieve this combination we have to take into consideration the characteristics of each component.

As a running example, let us consider the following rule set:

r1 : bird(X) $\rightarrow$ animal(X)   r2 : bird(X) $\Rightarrow$ flies(X)
r3 : brokenWing(X) $\Rightarrow$ $\neg$flies(X)   r3 $>$ r2

In this simple example we try to decide whether something is an animal and whether it is flying or not. Given the facts *bird(eagle)* and *brokenWing(eagle)*, as well as the superiority relation, we conclude that *animal(eagle)* and *¬flies(eagle)*.

Taking into account the fact that all predicates have only one argument, we can group together facts with the same argument value (using Map) and perform reasoning for each value separately (using Reduce). Pseudo-code for Map and $Reduce$ functions is depicted in Algorithm 2. Equivalently, we can view this process as performing reasoning on the rule set:

r1 : bird $\rightarrow$ animal   r2 : bird $\Rightarrow$ flies
r3 : brokenWing $\Rightarrow$ $\neg$flies   r3 $>$ r2

for each unique argument value.

Algorithm 2 single-argument inference

```
map(Long key, String value) :
    // key: position in document (irrelevant)
    // value: document line (a fact)
    argumentValue = extractArgumentValue(value);
    predicate = extractPredicate(value);
    EmitIntermediate(argumentValue, predicate);

reduce(String key, Iterator values) :
    // key: argument value
    // values : list of predicates (facts)
    List listOfFacts;
    Reasoner reasoner = Reasoner.getCopy();
    for each v in values
        listOfFacts.add(v);
    reasoner.Reason(listOfFacts);
    Emit(key , reasoner.getResults());
```

As far as MapReduce is concerned, Map function reads facts of the form *predicate(argumentValue)* and emits pairs of the form <argumentValue, predicate>.

Given the facts: *bird(eagle)*, *bird(owl)*, *bird(pigeon)*, *brokenWing(eagle)* and *brokenWing(owl)*, Map function will emit the following pairs :

<eagle, bird> <owl, bird> <pigeon, bird>
<eagle, brokenWing> <owl, brokenWing>

Then, reasoning is performed for each argument value (e.g., eagle, pigeon etc) separately, and in isolation. Therefore, the MapReduce framework will group/sort the pairs emitted by Map, resulting in the following pairs:

<eagle, <bird, brokenWing>>
<owl, <bird, brokenWing>> <pigeon, <bird>>

Reasoning is then performed during the reduce phase for each argument value in isolation, using the second rule set presented earlier (propositional form). For each $Reduce$ function, a copy of reasoner (described later on) gets as input a list of predicates and performs reasoning deriving and emitting new data. When all reduces are completed, the whole process is completed guaranteeing that every possible new data is inferred.

Returning to our example, the bullets below show the reasoning tasks that need to be performed. Note that each of these reasoning tasks can be performed in parallel with the others.

- *eagle* having *bird* and *brokenWing* as facts, deriving *animal(eagle)* and *¬flies(eagle)*
- *owl* having *bird* and *brokenWing* as facts, deriving *animal(owl)* and *¬flies(owl)*
- *pigeon* having *bird* as fact, deriving *animal(pigeon)* and *flies(pigeon)*.

For the purpose of conclusion derivation, we implemented a reasoner based on a variation of algorithm for propositional reasoning, described in ((Maher 2004)). Prior to any $Reduce$ function is applied, given rule set must be parsed initializing indexes and data structures required for reasoning. Although implementation details of the reasoner are out of the scope of this paper, we will explain all the functions used in Algorithm 2.

Each $Reduce$ function has to perform, in parallel, reasoning on the initial state of the reasoner. Thus, we use *Reasoner.getCopy()*, which provides a copy of the initialized reasoner. Subsequently, *reasoner.Reason(listOfFacts)* performs reasoning on each copy. In order to perform reasoning, *reasoner.Reason(listOfFacts)* gets as input the corresponding list of predicates (*listOfFacts*). Derived data are stored internally by each copy of the reasoner. The extraction of the derived data is performed by the *reasoner.getResults()*.

The algorithm for single-argument predicates is sound and complete since it performs reasoning using every given fact. This data partitioning does not alter resulting conclusions since facts with different argument values cannot produce conflicting literals and cannot be combined to reach new conclusions. Moreover, the reasoner is designed to derive all possible conclusions for each unique value. Thus, maximal and valid data derivation is assured.

4 Experimental results

We have implemented the algorithm for single-argument predicates in the Hadoop MapReduce framework[2], version 0.20. We have performed experiments on a cluster with 16 IBM System x iDataPlex nodes, using a Gigabit Ethernet interconnect. Each node was equipped with dual Intel Xeon Westmere 6-core processors, 128GB RAM and a single 1TB SATA hard drive.

4.1 Dataset

Due to no available benchmark, we generated our data set manually. In order to store facts directly to Hadoop Distributed File System (HDFS), facts were generated using MapReduce framework. We created a set of files consisting of generated pairs of the form <argumentValue, predicate>, with each pair corresponding to a unique fact. Finally, considering storage space, 1 billion facts correspond to 10 GB of data.

4.2 Rule set

To the best of our knowledge, there exist no standard defeasible logic rule set to evaluate our approach. For this reason, we decided to use synthetic data sets, namely the artificial rule set **teams(n)** appearing in ((Maher et al. 2001)). In **teams(n)** every literal is disputed, with $2^{(2*i)+1}$ rules of the form $a_{i+1} \Rightarrow a_i$ and $2^{(2*i)+1}$ rules of the form $a_{i+1} \Rightarrow \neg a_i$, for $0 \leq i \leq n$. The rules for a_i are superior to the rules for $\neg a_i$, resulting in $2^{(2*i)+1}$ superiority relations, for $0 \leq i \leq n$. For our experiments, we generated a **teams(n)** rule set for $n = 1$ which resulted in 20 defeasible rules, 10 superiority relations, 20 predicates appearing in the body of rules and 5 literals for conclusion derivation. This particular rule set was chosen because it is the only known benchmark for defeasible logics that involves "attacks".

4.3 Evaluation settings

We have evaluated our system in terms of the following parameters:

- **Runtime**, as the time required to calculate the inferential closure of the input, in minutes.
- **Number of nodes** performing the computation in parallel.
- **Dataset size**, expressed in the number of facts in the input.
- **Scaled speedup**, defined as $s = \frac{runtime_{1node} * N}{runtime_{Nnodes}}$, where $runtime_{1node}$ is the required run time for one node, N is the number of nodes and $runtime_{Nnodes}$ is the required run time for N nodes. It is a commonly used metric in parallel processing to measure how a system scales as the number of nodes increases. A system is said to scale sublinearly, superlinearly and linearly when $s < 1$, $s > 1$ and $s \simeq 1$ respectively.

4.4 Results

Figure 1 shows the runtime plotted against the size of the dataset, for various numbers of processing nodes and Figure 2 shows the scaled speedup for increasing number of

[2]http://hadoop.apache.org/mapreduce/

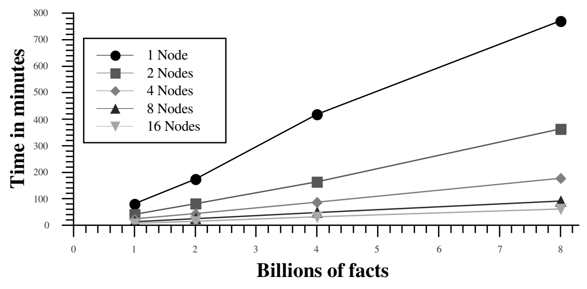

Figure 1: Runtime in minutes as a function of dataset size, for various numbers of nodes.

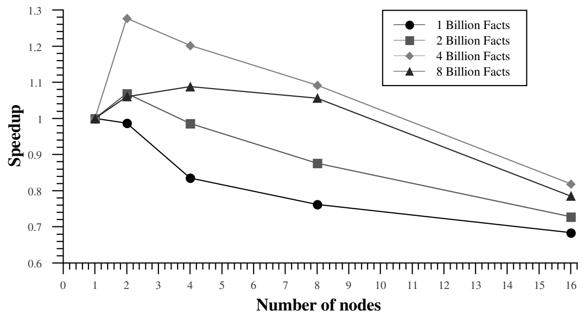

Figure 2: Scaled speedup for various dataset sizes.

nodes, and for various dataset sizes. Our results indicate the following:

- Our system easily scales to several billions of facts, even for a single node. In fact, we see no indication of the throughput decreasing as the size of the input increases.
- Our implementation demonstrates very high throughput (about 2,2 million facts per second) and is in league with state-of-the-art methods for monotonic logics ((Urbani et al. 2010)).
- Our system scales fairly linearly with the number of nodes. The loss in terms of scaled speedup for larger numbers of nodes and small datasets is attributed to platform overhead. Namely, starting a computational job in Hadoop incurs a significant computational overhead.
- In some cases, our system scales superlinearly. This is attributed to being able to store a larger part of the data in RAM. Although MapReduce relies on the hard drives for data transfer, the operating system uses RAM to improve disk access time and throughput, which explains the improved performance at some points.

In general, we attribute the demonstrated scalability to: (a) the limited communication required in our model, (b) the carefully designed load-balancing attributes of our algorithm, and, (c) the efficiency of Hadoop in handling large data volumes. To our best knowledge, this is the first work addressing nonmonotonic reasoning using mass parallelization techniques. Thus, we were unable to compare our findings with related work.

5 Conclusion and Future Work

This work is the first to explore the feasibility of nonmonotonic reasoning over huge data sets. We focused on simple nonmonotonic reasoning in the form of defeasible logic. We described how defeasible logic can be implemented in the MapReduce framework, and provided an extensive experimental evaluation for the case of reasoning with defeasible logic rules containing only single-argument predicates. Our results are very encouraging, and demonstrate that one can handle billions of facts using our approach.

We consider this to be just the starting point of a research effort towards supporting scalable parallel reasoning. We have been also considering the case of multi-argument predicates, and already have some encouraging results, which are not presented in this paper due to space limitations. As a next step, we intend to test its efficiency, as well as to study the effect of introducing increasingly complex defeasible rule sets, like those used in ((Maher et al. 2001)). Our expectation is that this case will also turn out to be fully feasible. Once achieved, this will allow the support of RDFS[3] reasoning and the execution of experiments over the Linked Open Data cloud[4]. In the longer term, we intend to apply the MapReduce framework to more complex knowledge representation methods, including Answer-Set programming ((Gelfond 2008)), RDF/S ontology evolution ((Konstantinidis et al. 2008)) and repair ((Roussakis, Flouris, and Christophides 2011)).

6 Acknowledgments

This work was partially supported by the PlanetData NoE (FP7:ICT-2009.3.4, #257641). We would like to thank Kostas Katrinis and Yoonseo Choi from the Exascale Systems research group at IBM for facilitating our experiments on the cluster.

References

Antoniou, G., and van Harmelen, F. 2008. *A Semantic Web Primer, 2nd Edition*. The MIT Press, 2 edition.

Bizer, C.; Heath, T.; and Berners-Lee, T. 2009. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.* 5(3):1–22.

Dean, J., and Ghemawat, S. 2004. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, 10–10. Berkeley, CA, USA: USENIX Association.

Gelfond, M. 2008. Chapter 7 answer sets. In Frank van Harmelen, V. L., and Porter, B., eds., *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*. Elsevier. 285 – 316.

Goodman, E. L.; Jimenez, E.; Mizell, D.; Al-Saffar, S.; Adolf, B.; and Haglin, D. J. 2011. High-performance computing applied to semantic databases. In *ESWC (2)*, 31–45.

Konstantinidis, G.; Flouris, G.; Antoniou, G.; and Christophides, V. 2008. A formal approach for rdf/s ontology evolution. In *Proceedings of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, 70–74. Amsterdam, The Netherlands, The Netherlands: IOS Press.

Kotoulas, S.; Oren, E.; and van Harmelen, F. 2010. Mind the data skew: distributed inferencing by speeddating in elastic regions. In Rappa, M.; Jones, P.; Freire, J.; and Chakrabarti, S., eds., *WWW*, 531–540. ACM.

Maher, M. J.; Rock, A.; Antoniou, G.; Billington, D.; and Miller, T. 2001. Efficient defeasible reasoning systems. *International Journal of Artificial Intelligence Tools* 10:2001.

Maher, M. J. 2004. Propositional defeasible logic has linear complexity. *CoRR* cs.AI/0405090.

Maluszynski, J., and Szalas, A. 2010. Living with inconsistency and taming nonmonotonicity. In *Datalog*, 384–398.

Mutharaju, R.; Maier, F.; and Hitzler, P. 2010. A mapreduce algorithm for el+. In Haarslev, V.; Toman, D.; and Weddell, G. E., eds., *Description Logics*, volume 573 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Oren, E.; Kotoulas, S.; Anadiotis, G.; Siebes, R.; ten Teije, A.; and van Harmelen, F. 2009. Marvin: Distributed reasoning over large-scale semantic web data. *J. Web Sem.* 7(4):305–316.

Roussakis, Y.; Flouris, G.; and Christophides, V. 2011. Declarative repairing policies for curated kbs. In *Proceedings of the 10th Hellenic Data Management Symposium (HDMS-11)*.

Urbani, J.; Kotoulas, S.; Oren, E.; and van Harmelen, F. 2009. Scalable distributed reasoning using mapreduce. In Bernstein, A.; Karger, D. R.; Heath, T.; Feigenbaum, L.; Maynard, D.; Motta, E.; and Thirunarayan, K., eds., *International Semantic Web Conference*, volume 5823 of *Lecture Notes in Computer Science*, 634–649. Springer.

Urbani, J.; Kotoulas, S.; Maassen, J.; van Harmelen, F.; and Bal, H. E. 2010. Owl reasoning with webpie: Calculating the closure of 100 billion triples. In Aroyo, L.; Antoniou, G.; Hyvönen, E.; ten Teije, A.; Stuckenschmidt, H.; Cabral, L.; and Tudorache, T., eds., *ESWC (1)*, volume 6088 of *Lecture Notes in Computer Science*, 213–227. Springer.

[3]http://www.w3.org/TR/rdf-schema/

[4]http://linkeddata.org/