Large-scale Parallel Stratified Defeasible Reasoning

Ilias Tachmazidis[1,2] and **Grigoris Antoniou**[1,3] and **Giorgos Flouris**[1] and **Spyros Kotoulas**[4] and **Lee McCluskey**[3]

Abstract. We are recently experiencing an unprecedented explosion of available data from the Web, sensors readings, scientific databases, government authorities 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 huge amounts of data for these applications. In this paper, we consider inconsistency-tolerant reasoning in the form of defeasible logic, and analyze how parallelization, using the MapReduce framework, can be used to reason with defeasible rules over huge datasets. We extend previous work by dealing with predicates of arbitrary arity, under the assumption of stratification. Moving from unary to multi-arity predicates is a decisive step towards practical applications, e.g. reasoning with linked open (RDF) data. Our experimental results demonstrate that defeasible reasoning with millions of data is performant, and has the potential to scale to billions of facts.

1 Introduction

Currently, we experience a significant growth of the amount of available data originating from sensor readings, scientific databases, government authorities etc. Such data are mainly published on the Web, providing easier knowledge exchange and interlinkage [21]. This yields the need for large and interconnected data, as shown by the Linked Open Data initiative[5] [5].

The study of knowledge representation has been mainly targeted on complex knowledge structures and reasoning methods for processing such structures. This raises the question whether such reasoning methods can be applied on huge datasets. Reasoning should be performed using rule sets that would allow the aggregation, visualization, understanding and exploitation of given datasets and their interconnections. Specifically, one should use rules able to encode inference semantics, commonsense and practical conclusions in order to infer new and useful knowledge based on the data. This is usually a formidable task when it comes to web-scale data: for example, as described in [24] for 78,8 million statements crawled from the Web, the number of inferred conclusions (RDFS closure) consists of 1,5 billion triples.

In this work, we study nonmonotonic rule sets [2], [18], which are suitable for encoding commonsense knowledge and reasoning, and providing supplementary advantages. Particularly, in case of poor quality data, the use of such rule sets can prevent triviality of inference. The occurrence of low quality data is common when they are fetched from different sources, which are not controlled by the data engineer.

Over the last years, parallel reasoning has been studied extensively e.g., in [20], [24], [12], [10] scaling reasoning up to 100 billion triples [23]. It could be addressed by the use of parallel reasoning techniques that would allow simultaneous processing over distinct chunks of data, with each chunk being assigned to a computer in the cloud.

Parallel reasoning can be based either on rule partitioning or on data partitioning [15]. Rule partitioning assigns the computation of each rule to a computer in the cloud. However, balanced work distribution in this case is difficult to achieve, as the computational burden per rule depends on the structure of the rule set. On the other hand, data partitioning assigns a subset of data to each computer in the cloud. Data partitioning is more flexible, providing more fine-grained partitioning and allowing easier distribution among nodes in a balanced manner.

Current approaches have focused on monotonic reasoning, such as RDFS and OWL-horst, or have not performed scalability evaluation [19]. Our paper deals with nonmonotonic approaches, and is therefore novel. Nonomonotonic reasoning has been chosen because it allows to overcome triviality of reasoning caused by inconsistent or incomplete data.

In particular, we consider defeasible rules and reasoning, and examine how nonmonotonic (defeasible) reasoning over huge datasets can be performed using massively parallel computational techniques. We adopt the MapReduce framework [6], which is widely used for parallel processing of huge datasets.

Previous work [22] described how defeasible logic with unary predicates can be implemented with MapReduce. Here we extend that work by considering predicates of arbitrary arity. From the applicability perspective, this is a decisive step, as most real-world data require multi-argument predicates. In particular, it opens the possibility of reasoning with semi-structured data, e.g. linked data expressed in RDF, where binary predicates are needed to express properties. From the technical perspective, reasoning with MapReduce turns out to be far more difficult, requiring multiple passes instead of the single-pass approach of [22]. In fact, for reasons explained later, our solution works under the requirement that the defeasible theory is stratified.

The paper is organized as follows. Section 2 introduces briefly MapReduce Framework and Defeasible Logic. An algorithm for multi-argument defeasible logic is described in Section 3, while experimental results are presented in Section 4. We conclude in Section 5.

[1] Institute of Computer Science, FORTH
[2] Department of Computer Science, University of Crete
[3] University of Huddersfield, UK
[4] IBM Research, IBM Ireland
[5] http://linkeddata.org/

2 Preliminaries

2.1 MapReduce Framework

MapReduce is a framework for parallel processing over huge datasets [6]. Processing is carried out in two phases, 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. 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.

Probably the most well known MapReduce example is the *word-count* example. In this example, we take as input a large number of documents and the final result is the calculation of the number of occurrences of each word. The pseudo-code for the Map and $Reduce$ functions is depicted in Algorithm below.

```
map(Long key, String value):
 // key: position in document (ignored)
 // 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 ("1"). Here we do not use the position of each line in the document, thus the *key* in Map is ignored. However, a word can be found more than once in a line. In this case we emit a <w, 1> pair for each occurrence. Consider the line "Hello world. Hello MapReduce.". Instead of emitting a pair <Hello, 2>, our simple example emits <Hello, 1> twice (pairs for words *world* and *MapReduce* are emitted as well). 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 final number of occurrences. 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 $\rightsquigarrow$. Strict rules are rules in the classical sense: whenever the premises are indisputable (e.g., facts) then so is the conclusion. Defeasible rules are rules that can be defeated by contrary evidence. Defeaters are rules that cannot be used to draw any conclusions; their only use is to prevent some conclusions.

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, meaning that q is definitely provable in D.
- $-\Delta$q, meaning that we have proved that q is not definitely provable in D.
- $+\partial$q, meaning that q is defeasibly provable in D.
- $-\partial$q, meaning 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 conditions shown below. 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. For more details on provability and an explanation of the intuition behind the conditions below, see [17].

$+\Delta$: 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

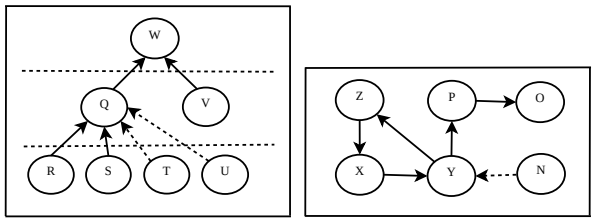
For reasons that will be explained later, defeasible reasoning over rule sets with multi-argument predicates is based on the dependencies between predicates which is encoded using the *predicate dependency graph*. Thus, rule sets can be divided into two categories: *stratified* and *non-stratified*. Intuitively, a *stratified* rule set can be represented as a hierarchy of dependencies between predicates, while a *non-stratified* not. We address the problem for stratified rule sets by

providing a well-defined reasoning sequence, and explain at the end of the section the challenges for non-stratified rule sets.

The dependencies between predicates can be represented using a *predicate dependency graph*. For a given rule set, the *predicate dependency graph* is a directed graph whose:

- vertices correspond to predicates. For each literal p, both p and $\neg p$ are represented by the positive predicate.
- edges are directed from a predicate that belongs to the body of a rule, to a predicate that belongs to the head of the same rule. Edges are used for all three rule types (strict rules, defeasible rules, defeaters). For a predicate p, edges with solid lines correspond to rules that support p while edges with dotted lines correspond to rules that support ¬p.

The superiority relation is not part of the graph. Figure 1a depicts a stratified rule set, while Figure 1b depicts a non-stratified rule set.



(a) Stratified (b) Non-stratified

Figure 1: Predicate dependency graph

Stratified rule sets (correspondingly, *non-stratified rule sets*) are rule sets whose predicate dependency graph is acyclic (correspondingly, contains a cycle). *Stratified theories* are theories based on stratified rule sets.

As an example of a stratified rule set, consider the following rule set:

r1: R(X,Z), S(Z,Y) ⇒ Q(X,Y).
r2: T(X,Z), U(Z,Y) ⇒ ¬Q(X,Y).
r3: Q(X,Y), V(Y,Z) ⇒ W(X,Z).
r1 > r2.

The predicate dependency graph for the above rule set is depicted in Figure 1a. The predicate graph can be used to determine strata for the different predicates. In particular, predicates (nodes) with no outgoing edges are assigned the maximum stratum, which is equal to the maximum depth of the directed acyclic graph (i.e., the size of the maximum path that can be defined through its edges), say k. Then, all predicates that are connected with a predicate of stratum k are assigned stratum $k-1$, and the process continues recursively until all predicates have been assigned some stratum. Note that predicates are reassigned to a lower stratum in case of multiple dependencies. The dashed horizontal lines in Figure 1a are used to separate the various strata, which, in our example, are as follows:

Stratum 2 : W
Stratum 1 : Q, V
Stratum 0 : R, S, T, U

A defeasible theory D is called *decisive* iff the predicate dependency graph of D is acyclic.

Proposition 1. *[4] If D is decisive, then for each literal p:*
(a) either $D \vdash +\Delta p$ or $D \vdash -\Delta p$
(b) either $D \vdash +\partial p$ or $D \vdash -\partial p$

Taking into consideration that stratified rule sets correspond to acyclic predicate dependency graphs, we can deduce using Proposition 1 that stratified rule sets provide decisive defeasible theories. Thus, for stratified rule sets, each literal p is:
(a) either $+\Delta$p or $-\Delta$p
(b) either $+\partial$p or $-\partial$p

In general, there are three possible states for each literal p:
(a) $+\Delta$p and $+\partial$p
(b) $-\Delta$p and $+\partial$p
(c) $-\Delta$p and $-\partial$p

Reasoning is based on facts. According to defeasible logic algorithm, facts are $+\Delta$ and every literal that is $+\Delta$, is $+\partial$ too. Having $+\Delta$ and $+\partial$ in our initial knowledge base, it is convenient to store data and perform reasoning only for $+\Delta$ and $+\partial$ predicates.

This representation of knowledge allows us to reason and store more efficiently provability information regarding various facts. In particular, if a literal is not found as a $+\Delta$ (correspondingly, $+\partial$) then it is $-\Delta$ (correspondingly, $-\partial$). In addition, stratified defeasible theories have the property that if we have computed all the $+\Delta$ and $+\partial$ conclusions up to a certain stratum, and a rule whose body contains facts of said stratum does not currently fire, then this rule will also be inapplicable in subsequent passes; this provides a well-defined reasoning sequence, namely considering rules from lower to higher strata.

3.1 Reasoning overview

During reasoning we will use the representation (<fact, ($+\Delta$,$+\partial$)>) to store our inferred facts. We begin by transforming the given facts into (<fact, ($+\Delta$,$+\partial$)>). This transformation is carried out in a single MapReduce pass.

Now let's consider for example the facts *R(a,b)*, *S(b,b)*, *T(a,e)*, *U(e,b)* and *V(b,c)*. The initial pass on these facts using the aforementioned rule set will create the following output :

<R(a,b), ($+\Delta$,$+\partial$)> <S(b,b), ($+\Delta$,$+\partial$)>
<T(a,e), ($+\Delta$,$+\partial$)> <U(e,b), ($+\Delta$,$+\partial$)>
<V(b,c), ($+\Delta$,$+\partial$)>

No reasoning needs to be performed for the lowest stratum (stratum 0) since these predicates (R,S,T,U) do not belong to the head of any rule. As is obvious by the definition of $+\partial$, $-\partial$, defeasible logic introduces uncertainty regarding inference, because certain facts/rules may "block" the firing of other rules. As explained above, this can be prevented if we reason for each stratum separately, starting from the lowest stratum and continuing to higher strata. This is the reason why for a hierarchy of N strata we have to perform $N-1$ times the procedure described below. In order to perform defeasible reasoning we have to run two passes for each stratum. The first pass computes which rules can fire. The second pass performs the actual reasoning and computes for each literal if it is definitely or defeasibly provable. The reasons for both decisions (reasoning sequence and two passes per strata) are explained in the end of the next subsection.

3.2 Pass #1: Fired rules calculation

As far as the first pass is concerned, we calculate the inference of fired rules based on techniques used for basic and multi-way join as described in [8] and [1]. Basic join is performed on common argument values. Consider the following rule :

r1: R(X,Z), S(Z,Y) ⇒ Q(X,Y).

The key observation is that relations R and S can be joined on their common argument Z. Based on this observation, during Map operation we emit pairs of the form <Z,(X,R)> for predicate R and <Z,(Y,S)> for predicate S. The idea is to join R and S only for literals that have the same value on argument Z. During $Reduce$ operation we combine R and S producing Q.

In our example, the facts *R(a,b)* and *S(b,b)* will cause Map to emit <b,(a,R)> and <b,(b,S)>. MapReduce framework groups and sorts intermediate pairs passing <b,<(a,R),(b,S)>> to $Reduce$ operation. Finally, at $Reduce$ we combine given values and infer Q(a,b).

To support defeasible logic rules which have blocking rules, this approach must be extended. We must record all fired rules prior to any conclusion inference, whereas for monotonic logics this is not necessary, and conclusion derivation can be performed immediately. The reason why this is so is explained at the end of the subsection. Pseudo-code for Map and $Reduce$ functions, for basic join, is depicted in Algorithm below. Map function reads input of the form <literal, ($+\Delta$,$+\partial$)> or <literal, ($+\partial$)> and emits pairs of the form <matchingArgumentValue, (nonMatchingArgumentValue, Predicate, $+\Delta$, $+\partial$)> or <matchingArgumentValue, (nonMatchingArgumentValue, Predicate, $+\partial$)> respectively.

```
map(Long key, String value):
 // key: position in document (irrelevant)
 // value: document line (derived conclusion)
 For every common argumentValue in value
  EmitIntermediate(argumentValue, value);

reduce(String key, Iterator values):
 // key: matching argument
 // value: literals for matching
 For every argument value match in values
  If Strict rule fired with +Δ premises then
   Emit(firedLiteral, "+Δ, +∂, ruleID");
  else
   Emit(firedLiteral, "+∂, ruleID");
```

Now consider again the stratified rule set described in the beginning of the section, for which the initial pass will produce the following output:

<R(a,b), ($+\Delta$,$+\partial$)> <S(b,b), ($+\Delta$,$+\partial$)>
<T(a,e), ($+\Delta$,$+\partial$)> <U(e,b), ($+\Delta$,$+\partial$)>
<V(b,c), ($+\Delta$,$+\partial$)>

We perform reasoning for stratum 1, so we will use as premises all the available information for predicates of stratum 0. Map function will emit the following pairs :

<b, (a,R,$+\Delta$,$+\partial$)> <b, (b,S,$+\Delta$,$+\partial$)>
<e, (a,T,$+\Delta$,$+\partial$)> <e, (b,U,$+\Delta$,$+\partial$)>
<b, (c,V,$+\Delta$,$+\partial$)>

MapReduce framework will perform grouping/sorting resulting in the following intermediate pairs :

<b, <(a,R,$+\Delta$,$+\partial$), (b,S,$+\Delta$,$+\partial$), (c,V,$+\Delta$,$+\partial$)>>
<e, <(a,T,$+\Delta$,$+\partial$), (b,U,$+\Delta$,$+\partial$)>>

During reduce we combine premises in order to emit the *firedLiteral* which consists of the fired rule head predicate and the *nonMatchingArgumentValue* of the premises. However, inference depends on the type of the rule. In general for all three rule types (strict rules, defeasible rules and defeaters) if a rule fires then we emit as output <firedLiteral, ($+\partial$,ruleID)>. However, there is a special case for strict rules. This special case covers the required information for $+\Delta$ conclusions inference. If all premises are $+\Delta$ then we emit as output <firedLiteral, ($+\Delta$,$+\partial$,ruleID)> instead of <firedLiteral, ($+\partial$, ruleID)>.

For example during the reduce phase the reducer with key :

b will emit <Q(a,b), ($+\partial$, r1)>
e will emit <¬Q(a,b), ($+\partial$, r2)>

As we see here, *Q(a,b)* and *¬Q(a,b)* are computed by different reducers which do not communicate with each other. Thus, none of the two reducers have all the available information in order to perform defeasible reasoning. Therefore, we need a second pass for the reasoning.

Let us illustrate why reasoning has to be performed for each stratum separately. Consider again our running example. During the reduce phase we cannot join *Q(a,b)* with *V(b,c)* because we do not have a final conclusion on *Q(a,b)*. Thus, we will not perform reasoning for *W(a,c)* during the second pass, which leads to data loss. However, if another rule (say r4) supporting *¬W(a,c)* had also fired, then during the second pass, we would have mistakenly inferred *¬W(a,c)*, leading our knowledge base to inconsistency.

3.3 Pass #2: Defeasible reasoning

We proceed with the second pass. Once fired rules are calculated, a second MapReduce pass performs reasoning for each literal separately. We should take into consideration that each literal being processed, could already exist in our knowledge base (due to the initial pass). In this case we perform a duplicate elimination by not emitting pairs for existing conclusions. Pseudo-code for Map and $Reduce$ functions, for stratified rule sets, is depicted in Algorithm below.

```
map(Long key, String value) :
 // key: position in document (irrelevant)
 // value: inferred knowledge/fired rules
 String p = extractLiteral(value);
 String knowledge = extractKnowledge(value);
 If p starts with "¬"
  p = eliminateComplementarySign(p);
  knowledge = "¬" + knowledge;
 EmitIntermediate(p, knowledge);

reduce(String p, Iterator values) :
 // p: a literal
 // values : inferred knowledge/fired rules
 For each value in values
  markKnowledge(value);
 For literal in {p, ¬p} check
  If literal is already +Δ then
   Return;
  Else if Strict rule with +Δ premises then
   Emit(literal, +"Δ, +∂");
  Else If literal is +∂ after reasoning then
    Emit(literal, +"∂");
```

After both initial pass and rule inference (first pass), our knowledge will consists of:

<R(a,b), ($+\Delta$,$+\partial$)> <S(b,b), ($+\Delta$,$+\partial$)>
<T(a,e), ($+\Delta$,$+\partial$)> <U(e,b), ($+\Delta$,$+\partial$)>
<V(b,c), ($+\Delta$,$+\partial$)> <Q(a,b), ($+\partial$, r1)>
<¬Q(a,b), ($+\partial$, r2)>

During Map operation we must first extract from value the literal and the inferred knowledge or the fired rule using *extractLiteral()* and *extractKnowledge()* respectively. For each literal p we must send both p and $\neg p$ to the same reducer. This is achieved by eliminating "¬" from p (using *eliminateComplementarySign()*) and adding it to the $knowledge$ (where $knowledge$ contains inferred knowledge or fired rule). The Map function will emit the following pairs :

<R(a,b), (+Δ,+∂)> <S(b,b), (+Δ,+∂)>
<T(a,e), (+Δ,+∂)> <U(e,b), (+Δ,+∂)>
<V(b,c), (+Δ,+∂)> <Q(a,b), (+∂, r1)>
<Q(a,b), (¬, +∂, r2)>

MapReduce framework will perform grouping/sorting resulting in the following intermediate pairs :

<R(a,b), (+Δ,+∂)> <S(b,b), (+Δ,+∂)>
<T(a,e), (+Δ,+∂)> <U(e,b), (+Δ,+∂)>
<V(b,c), (+Δ,+∂)> <Q(a,b), <(+∂, r1), (¬,+∂, r2)>>

At $Reduce$ the key contains the literal and the values contain all the available information for that literal (known knowledge, fired rules). We traverse over *values* marking known knowledge and fired rules using the *markKnowledge()* function. Subsequently, we use this information in order to perform reasoning for each literal.

During the reduce phase the reducer with key :

R(a,b), *S(b,b)*, *T(a,e)*, *U(e,b)*, *S(b,c)*, *V(b,c)* will not emit anything
Q(a,b) will emit <Q(a,b), (+∂)>

Literals *R(a,b)*, *S(b,b)*, *T(a,e)*, *U(e,b)*, *S(b,c)* and *V(b,c)* are known knowledge. For known knowledge a potential duplicate elimination must be performed. We reason simultaneously both for Q(a,b) and ¬Q(a,b). As ¬Q(a,b) is $-\partial$, it does not need to be recorded. Note that duplicate elimination affects the parallelization. This issue is discussed in Section 4. Additionally, highly skewed datasets result in literals with highly skewed amounts of corresponding knowledge, decreasing the overall parallelization. We address this issue by eliminating identical knowledge between map and reduce phase.

3.4 Final remarks

In order to perform sound and complete reasoning for stratified rule sets we need $2*(N-1)+1$ MapReduce passes (N being the number of strata). The total number of MapReduce passes is independent of the given input. As mentioned above, performing reasoning for each stratum separately ensures that every possible conclusion is indeed inferred. The soundness of the whole approach is based on our initial decision on which conclusions we actually need to store. Eventually, our knowledge base consists of $+\Delta$ and $+\partial$ literals.

The situation for non-stratified rule sets is more complex. Reasoning can be based on the algorithm described in [16], performing reasoning until no new conclusion is derived. Hence, the total number of required passes is generally unpredictable, depending both on given rule set and data distribution. Additionally, an efficient mechanism for "$\forall$r $\in$ R$_s$[q] $\exists\alpha \in$ A(r): $-\Delta\alpha \in$ P(1..i)" and "$\forall$r $\in$ R$_{sd}$[q] $\exists\alpha \in$ A(r): $-\partial\alpha \in$ P(1..i)" computation is yet to be defined. Finally, we have to reason for and store every possible conclusion ($+\Delta$,$-\Delta$,$+\partial$,$-\partial$), producing a significantly larger stored knowledge base.

4 Experimental results

We have implemented our method using Hadoop, and provide experimental results on a cluster. We have evaluated our system in terms of its ability to handle large data files, its scalability with the number of maching and its scalability with regard to the number of rules in each stratum.

4.1 Dataset

Due to no available benchmark, we based our experiments on manually generated datasets. The generated dataset consists of a set of $+\Delta$ literals. Each literal is represented either as "*predicate(argumentValue)* $+\Delta$" or as "*predicate(argumentValue, argumentValue)* $+\Delta$". In order to simulate a real-world dataset, we used statistics on semantic web data. As described in [13] and [7], semantic web data are highly skewed following zipf distribution. Thus, we used a zipf distribution generator in order to create a dataset resembling real-world datasets. For our experiments, we generated a total of 500 million facts corresponding to 10 GB of data.

4.2 Rule set

To the best of our knowledge, there exists no standard defeasible logic rule set to evaluate our approach. For evaluation purposes, taking into consideration rule sets appearing in [16], we created an artificial rule set named **blocking(n)**. In **blocking(n)** there are $n/2$ rules of the form "Q$_i$(X), R$_i$(X,Y) $\Rightarrow$ Q$_{i+1}$(Y)" and $n/2$ of the form "Q$_i$(X), S$_i$(X,Y) $\Rightarrow$ ¬Q$_{i+1}$(Y)". Rules supporting Q_{i+1}*(Y)* are superior to rules supporting $\neg Q_{i+1}$*(Y)*, resulting $n/2$ superiority relations. For experimental results we used **blocking(n)** for $n = \{2, 4, 8, 16\}$.

4.3 Platform

We have experimented on a cluster of virtual machines on an IBM Cloud, running IBM Hadoop Cluster v1.3, which is compatible with Apache Hadoop v20.2. Each node was equipped with a single CPU core, 1GB of main memory and 55GB of hard disk space. We have scaled the number of nodes from 1 to 16, using a single master node.

4.4 Results

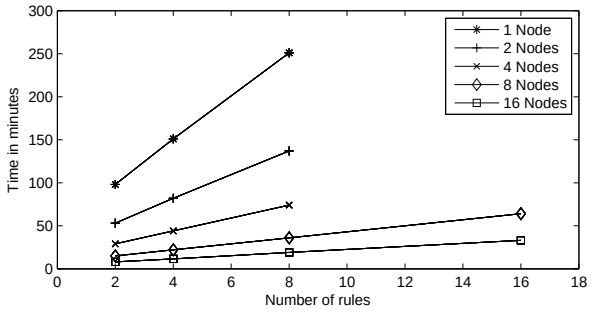


Figure 2: Runtime in minutes as a function of number of rules, for various numbers of nodes.

Figure 2 shows the scaling properties of our system for 2, 4 and 8 rules, there was not enough hard disk space to run all the experiments with 16 rules. We observe the following: (i) even in our modest setup,

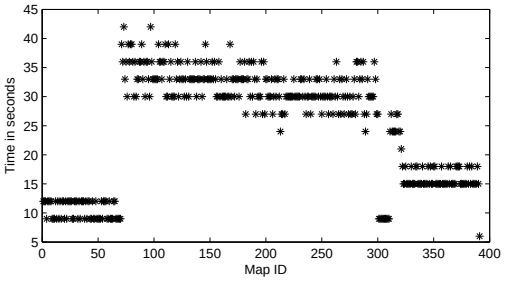


Figure 3: Time in seconds for each map during the second pass

the runtime is in kept very low, considering the size of the knowledge base, (ii) our system scales linearly with the number of rules; the runtime roughly doubles for twice the number of rules, (iii) our system scales linearly with the number of nodes; the runtime halves when we double the number of nodes.

The above show that our system is indeed capable of achieving high performance and scales very well, both with regard to the number of nodes and the number of rules. Nevertheless, to further investigate how our system would perform beyond this, it is critical to examine the load-balancing properties of our algorithm, a major scalability barrier in parallel applications in this domain [14]. Figure 3 show the load balance between different tasks for 16 nodes on 8 rules. In principle, an application performs badly when a single task dominates the runtime, since all other tasks would need to wait for it to finish. In our experiments, it is obvious that no such task exists.

5 CONCLUSION

In this paper we extended previous work described in [22] by proposing a method to perform reasoning for multi-argument predicates under the assumption of stratification. Multi-argument predicates complicate significantly the implementation (compared to the one presented in [21] for single-argument predicates), because multi-argument predicates require multiple passes. We presented how reasoning can be implemented in the MapReduce framework and provided an extensive experimental evaluation. The results demonstrate that our approach can address reasoning over millions of facts.

We consider this to be another step in the research effort towards supporting scalable parallel reasoning. Subsequently, we intend to support defeasible logic for multi-argument predicates without the stratification assumption. In the longer term, we intend to apply the MapReduce framework to more complex knowledge representation methods, including Answer-Set programming [9], systems of argumentation [3], and RDF/S ontology evolution [11] and repair [21].

Another potential area that may benefit from this work is AI Planning. Here the most competitive modern planning engines work by first grounding planning operators (which in general are made up of non-unary predicates) into thousands of ground operators called actions. The pre- and post-condition structure of these actions gives rise to planning dependency graphs also manifested as *causal graphs* [Helmert 06]. Given the non-monotonic nature of planning, in our future work we plan to apply the techniques developed here to help stratify large action databases, in order to leverage massively parallel computation to speed up goal achievement and hence plan construction.

ACKNOWLEDGEMENTS

This work was partially supported by the PlanetData NoE (FP7:ICT-2009.3.4, #257641).

REFERENCES

[1] Foto N. Afrati and Jeffrey D. Ullman, 'Optimizing joins in a mapreduce environment', in *In EDBT*, (2010).
[2] Grigoris Antoniou and Frank van Harmelen, *A Semantic Web Primer, 2nd Edition*, The MIT Press, 2 edn., March 2008.
[3] Philippe Besnard and Anthony Hunter, *Elements of Argumentation*, The MIT Press, 2008.
[4] David Billington, 'Defeasible logic is stable', *J. Log. Comput.*, **3**(4), 379–400, (1993).
[5] Christian Bizer, Tom Heath, and Tim Berners-Lee, 'Linked data - the story so far', *Int. J. Semantic Web Inf. Syst.*, **5**(3), 1–22, (2009).
[6] Jeffrey Dean and Sanjay Ghemawat, 'Mapreduce: simplified data processing on large clusters', in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, pp. 10–10, Berkeley, CA, USA, (2004). USENIX Association.
[7] Songyun Duan, Anastasios Kementsietsidis, Kavitha Srinivas, and Octavian Udrea, 'Apples and oranges: a comparison of rdf benchmarks and real rdf datasets', in *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pp. 145–156, New York, NY, USA, (2011). ACM.
[8] Florian Fische, 'Investigation & design for rule-based reasoning', Technical report, LarKC, (2010).
[9] Michael Gelfond, 'Chapter 7 answer sets', in *Handbook of Knowledge Representation*, eds., Vladimir Lifschitz Frank van Harmelen and Bruce Porter, volume 3 of *Foundations of Artificial Intelligence*, 285 – 316, Elsevier, (2008).
[10] Eric L. Goodman, Edward Jimenez, David Mizell, Sinan Al-Saffar, Bob Adolf, and David J. Haglin, 'High-performance computing applied to semantic databases', in *ESWC (2)*, pp. 31–45, (2011).
[11] George Konstantinidis, Giorgos Flouris, Grigoris Antoniou, and Vassilis Christophides, 'A formal approach for rdf/s ontology evolution', in *Proceedings of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, pp. 70–74, Amsterdam, The Netherlands, The Netherlands, (2008). IOS Press.
[12] Spyros Kotoulas, Eyal Oren, and Frank van Harmelen, 'Mind the data skew: distributed inferencing by speeddating in elastic regions', in *WWW*, eds., Michael Rappa, Paul Jones, Juliana Freire, and Soumen Chakrabarti, pp. 531–540. ACM, (2010).
[13] Spyros Kotoulas, Eyal Oren, and Frank van Harmelen, 'Mind the data skew: Distributed inferencing by speeddating in elastic regions', in *Proceedings of the WWW 2010, Raleigh NC, USA*, (2010).
[14] Spyros Kotoulas, Eyal Oren, and Frank van Harmelen, 'Mind the data skew: distributed inferencing by speeddating in elastic regions', in *Proceedings of the 19th international conference on World wide web*, WWW '10, pp. 531–540, New York, NY, USA, (2010). ACM.
[15] Spyros Kotoulas, Frank van Harmelen, and Jesse Weaver, 'Kr and reasoning on the semantic web: Web-scale reasoning', in *Handbook of Semantic Web Technologies*, eds., John Domingue, Dieter Fensel, and James A. Hendler, 441–466, Springer Berlin Heidelberg, (2011). 10.1007/978-3-540-92913-0_11.
[16] M. J. Maher, A. Rock, G. Antoniou, D. Billington, and T. Miller, 'Efficient defeasible reasoning systems', *International Journal of Artificial Intelligence Tools*, **10**, 2001, (2001).
[17] Michael J. Maher, 'Propositional defeasible logic has linear complexity', *CoRR*, **cs.AI/0405090**, (2004).
[18] Jan Maluszynski and Andrzej Szalas, 'Living with inconsistency and taming nonmonotonicity', in *Datalog*, pp. 384–398, (2010).
[19] Raghava Mutharaju, Frederick Maier, and Pascal Hitzler, 'A mapreduce algorithm for el+', in *Description Logics*, eds., Volker Haarslev, David Toman, and Grant E. Weddell, volume 573 of *CEUR Workshop Proceedings*. CEUR-WS.org, (2010).
[20] Eyal Oren, Spyros Kotoulas, George Anadiotis, Ronny Siebes, Annette ten Teije, and Frank van Harmelen, 'Marvin: Distributed reasoning over large-scale semantic web data', *J. Web Sem.*, **7**(4), 305–316, (2009).
[21] Yannis Roussakis, Giorgos Flouris, and Vassilis Christophides, 'Declarative repairing policies for curated kbs', in *Proceedings of the 10th Hellenic Data Management Symposium (HDMS-11)*, (2011).

[22] Ilias Tachmazidis, Grigoris Antoniou, Giorgos Flouris, and Spyros Kotoulas, 'Towards parallel nonmonotonic reasoning with billions of facts', (2012).

[23] Jacopo Urbani, Spyros Kotoulas, Jason Maassen, Frank van Harmelen, and Henri E. Bal, 'Owl reasoning with webpie: Calculating the closure of 100 billion triples', in *ESWC (1)*, eds., Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, volume 6088 of *Lecture Notes in Computer Science*, pp. 213–227. Springer, (2010).

[24] Jacopo Urbani, Spyros Kotoulas, Eyal Oren, and Frank van Harmelen, 'Scalable distributed reasoning using mapreduce', in *International Semantic Web Conference*, eds., Abraham Bernstein, David R. Karger, Tom Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, volume 5823 of *Lecture Notes in Computer Science*, pp. 634–649. Springer, (2009).