Towards Parallel Nonmonotonic Reasoning with Billions of Facts

Ilias Tachmazidis 1,2, Grigoris Antoniou 2,3, Giorgos Flouris 2, Spyros Kotoulas 4

(tahmazid,antoniou, tgeo)@ics.forth.gr , Spyros.Kotoulas@ie.ibm.com

1 University of Crete
Department of Computer Science
http://www.csd.uoc.gr

2 Foundation for Research and Technology - Hellas,
Institute of Computer Science
http://www.ics.forth.gr

3 University of Huddersfield, UK
http://www.hud.ac.uk

4 IBM Research, Ireland
http://www.ibm.com

Motivation for this Paper

Situation: In the last two of years, there has been significant progress in parallel reasoning, scaling reasoning up to 100 billion triples. Nevertheless, current approaches have been restricted to monotonic reasoning, namely RDF and OWL-hor, or have not been evaluated for scalability.

Problems: Poor quality data (e.g., involving inconsistency or incompleteness), could easily lead to reasoning trivially when considering rules based on monotonic formalisms.

Purpose of this paper:
1. Analyze how parallelization, using the MapReduce framework, can be used to reason with defeasible rules over huge data sets.
2. Evaluate our approach in terms of scalability.

Deontic Logic

Syntax

Facts
e.g. bird(eagle)

Strict Rules
e.g. (bird(X) → animal(X))

Defeasible Rules
e.g. bird(X) ⇒ flies(X)

Defeaters
e.g. brokenWing(X) ⇒ ¬flies(X)

Priority Relation (acyclic relation on the set of rules)
e.g. r: bird(X) ⇒ flies(X)
r': brokenWing(X) ⇒ ¬flies(X)
r' > r

Advantages

Suitable for encoding commonsense knowledge and reasoning
Avoid triviality of inference due to low-quality data
Low complexity
The consequences of a defeasible theory D can be computed in O(N) time, where N is the number of symbols in D

MapReduce Framework

MapReduce is a software framework introduced by Google in 2004.
Deals with:
• very large amounts of data (many terabytes)
• process data fairly quickly
• use very large numbers of commodity machines (thousands)

Want an infrastructure that takes care of management tasks
• distribution of data
• management of fault tolerance
• collecting results
For a specific problem
• developer writes a few routines
• routines plug into the general interface

MapReduce Algorithm

Input:
Facts in multiple files

Spill Phase:
Grouping/Sorting

Apply Map Function

Reduce phase Input
Final Output

Rule set

(r1: bird(X) ⇒ animal(X),
r2: brokenWing(X) ⇒ ¬flies(X),
r3: brokenWing(eagle),
r4: brokenWing(owl))

Experimental Results

Runtime in minutes as a function of dataset size, for various numbers of nodes

Speedup

Scaled speedup for various dataset sizes