MASP-Reduce: A Proposal for Distributed Computation of Stable Models

Federico Igne
University of Udine, Udine, Italy and New Mexico State University, NM, USA
ignefederico@gmail.com

Agostino Dovier
University of Udine, Udine, Italy
agostino.dovier@uniud.it

Enrico Pontelli
New Mexico State University, NM, USA
epontell@cs.nmsu.edu

Abstract

There has been an increasing interest in recent years towards the development of efficient solvers for Answer Set Programming (ASP) and towards the application of ASP to solve increasing more challenging problems. In particular, several recent efforts have explored the issue of scalability of ASP solvers when addressing the challenges caused by the need to ground the program before resolution. This paper offers an alternative solution to this challenge, focused on the use of distributed programming techniques to reason about ASP programs whose grounding would be prohibitive for mainstream ASP solvers. The work builds on a proposal of a characterization of answer set solving as a form of non-standard graph coloring. The paper expands this characterization to include syntactic extensions used in modern ASP (e.g., choice rules, weight constraints). We present an implementation of the solver using a distributed programming framework specifically designed to manipulate very large graphs, as provided by Apache Spark, which in turn builds on the MapReduce programming framework. Finally, we provide a few preliminary results obtained from the first prototype implementation of this approach.

2012 ACM Subject Classification Computing methodologies → Logic programming and answer set programming, Software and its engineering → Massively parallel systems

Keywords and phrases ASP solving, Parallelism, Map-reduce

Digital Object Identifier 10.4230/OASIcs.ICLP.2018.8

Funding A. Dovier is partially supported by UNIUD PRID "Encase" and by INdAM GNCS projects. F. Igne benefited of a scholarship from the Scuola Superiore of the University of Udine.

Acknowledgements We thank Huiping Cao for the availability of the BigDat cluster (KDD lab).

The availability of efficient answer set solvers (e.g., CLASP and its descendants [8, 2]) gave Answer set programming (ASP) a leading role in languages for knowledge representation and reasoning. The simple syntax is surely one of the main strengths of the paradigm; moreover the stable models semantics intuitively resembles the human reasoning process in a clean and *logical* way. ASP is regarded as the computational embodiment of non-monotonic reasoning because of its simple syntax and elegant non-monotonic semantics. The popularity of ASP is demonstrated by the increasing number of authors publishing ASP-based research work in artificial intelligence as well as non-logic programming venues, and its use as a natural alternative to other paradigms (e.g., SAT solving). Most of the answer set solvers are currently developed as two-phases procedures (save some exceptions – e.g., [3, 11]) . The first stage is called *grounding* and computes the equivalent propositional logic program of an input logic program, instantiating each rule over the domain of its variables. Modern solvers also apply some simplifications and heuristics to the program, in order to ease the computation during the second stage. The computation of the answer sets of a logic program is carried out by the *solving* stage, which also deals with the non-deterministic reasoning involved in the model.

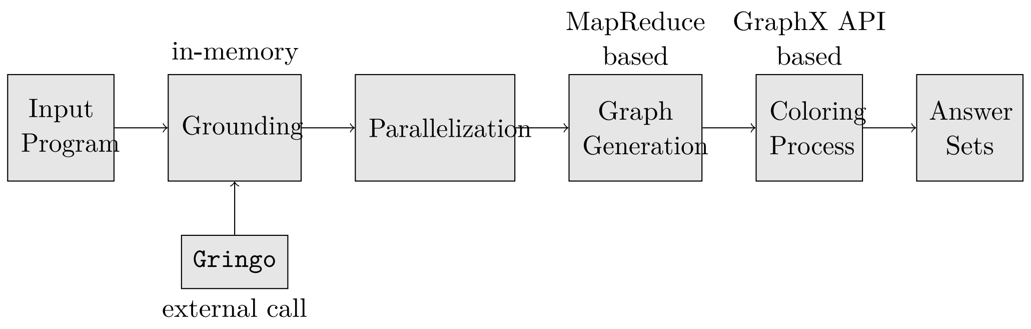
ASP encoding of sophisticated applications in real-world domains (e.g., planning, phylogenetic inference) highlighted the strengths and weaknesses of this paradigm. Most of the times, the technology underlying the ASP solvers, lacks the ability to keep up with the demand of complex applications. This has been, for example, highlighted in a study on the use of ASP to address complex planning problems [13, 5, 6]. With respect to these studies, it is clear that one of the main limitations of this paradigm resides in the grounding process and the ability to compute the stable models of large ground programs. This limitation is even more obvious when the whole computation is performed in-memory.

This work tries to partially solve the problem of processing large ground programs that can exceed capabilities for in-memory computation – using parallelism and distributed computing. We aim to study, analyse and develop a fully distributed answer set solver and use a distributed environment to efficiently represent and reason over large programs whose grounding would be prohibitive for a single general-purpose machine. We popose a solver that uses *MapReduce*, a distributed programming paradigm, mainly used to work with huge volumes of data on structured networks of computers (*workers*) [4]. Implementations of the MapReduce model (e.g., [4]) are usually executed on clusters to take full advantage of the parallel nature of the architecture. The paradigm provides a basic interface consisting of two methods: map(·) that maps a function over a collection of objects and outputs a collection of "key-value" tuples; reduce(·) that takes as input a collection of key-value pairs and merges the values of all entries with the same key; the merging operation is user-defined.

An inspiration for the work proposed here comes from the proposal by Konczak et al. [9, 10], which addresses the problem of finding the answer sets of a ground normal logic program by means of computing the admissible colorings of the relative Rule Dependency Graph (RDG). This is done by defining a set of operators on the RDG of a program. These operators deal with the non-deterministic coloring of nodes and the deterministic propagation of colors. [1] used this technique in the development of the NoMoRe (Non-monotonic Reasoning with Logic Programs) solver. This implementation is purely sequential and in-memory.

In this research we investigate the above-mentioned graph coloring approach and extended it so as to include weight constraint rules. We investigate its mapping to MapReduce and other distributed programming paradigms that build over MapReduce. The solver we are developing, called MASP-Reduce, is written in Scala [12, 7], it uses Apache Spark [14] as a library for distributed computation, and it natively works on the Hadoop Distributed File System (HDFS). The library gives access to a complete set of primitives for the *MapReduce* programming paradigm, and on top of this, it implements GraphX, a distributed direct multigraph with a complete and easy-to-use interface [14].

The development of *MASP-Reduce* is heavily based on the concept of rule dependency graph of ASP programs. Graphs turn out to be a good data structure for distributed programming, since they can directly exploit the underlying network configuration. Up to now, the software is comprehensive of a solver and of a graph generator that converts a ground program in a rule dependency graph (see Figure below). As a future work, we plan to implement a distributed grounder taking full advantage of the MapReduce paradigm, so that the Grounding block is incorporated into the Parallelization block.



We tested the solver both in a local environment (a notebook) and in a distributed environment, namely four nodes of a cluster, where each node is a 12-core Intel CPUs, with each core dual hyperthreaded for a total of 48 OS-visible processors per node; each node has 256GB of RAM, ∼3TB of hard disk local storage and ∼512GB solid state local storage. The implementation works on simple examples. However, during the development we encountered a few challenges that prevented us from providing a full testing phase report. Spark is presented as an easy and ready-to-use tool for distributed programming; this might be true in a few cases, but most of the times one needs to fine-tune the system in order to reach an optimal configuration; this tuning process takes into account a vast number of parameters, and is mostly program-specific – and it is work in progress for our project.

As far as we know, this is the first work addressing the implementation of a distributed answer set solver using MapReduce paradigm and non-standard graph coloring as answer set characterization. This deeply influenced own roadmap, which couldn't take advices from previous works, leading to an incremental approach to development.

The system is still far from complete; we are planning on working on the development of a distributed grounder in the next few months. We are also considering the implementation of a few coloring heuristics and learning techniques to improve the performances of the solver.

References

1 C. Anger, M. Gebser, T. Linke, A. Neumann, and T. Schaub. The NoMoRe++ System. In Chitta Baral, Gianluigi Greco, Nicola Leone, and Giorgio Terracina, editors, *Logic Programming and Nonmonotonic Reasoning, 8th International Conference, LPNMR 2005, Diamante, Italy, September 5-8, 2005, Proceedings*, volume 3662 of *LNCS*, pages 422–426. Springer, 2005.

2 Mutsunori Banbara, Benjamin Kaufmann, Max Ostrowski, and Torsten Schaub. Clingcon: The next generation. *TPLP*, 17(4):408–461, 2017. `doi:10.1017/S1471068417000138`.

3 A. Dal Palù, A. Dovier, E. Pontelli, and G. Rossi. GASP: Answer set programming with lazy grounding. *Fundam. Inform.*, 96(3):297–322, 2009.

4 J. Dean and S. Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*, volume 51, pages 107–113. ACM, January 2008.

5 Agostino Dovier, Andrea Formisano, and Enrico Pontelli. An empirical study of constraint logic programming and answer set programming solutions of combinatorial problems. *J. Exp. Theor. Artif. Intell.*, 21(2):79–121, 2009. `doi:10.1080/09528130701538174`.

6 Agostino Dovier, Andrea Formisano, and Enrico Pontelli. Perspectives on Logic-Based Approaches for Reasoning about Actions and Change. In Marcello Balduccini and Tran Cao

Son, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, volume 6565 of *Lecture Notes in Computer Science*, pages 259–279. Springer, 2011.

7 École Polytechnique Fédérale. Scala — Object-Oriented Meets Functional (website), 2018. [last accessed Feb. 2018] `http://www.scala-lang.org/`.

8 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo = ASP + Control: Preliminary Report. *CoRR*, abs/1405.3694, 2014. `arXiv:1405.3694`.

9 K. Konczak, T. Linke, and T. Schaub. Graphs and Colorings for Answer Set Programming: Abridged Report. In Vladimir Lifschitz and Ilkka Niemelä, editors, *Logic Programming and Nonmonotonic Reasoning: 7th International Conference, LPNMR 2004 Fort Lauderdale, FL, USA, January 6-8, 2004 Proceedings*, volume 2923 of *Lecture Notes in Computer Science*, pages 127–140. Springer, 2004.

10 Kathrin Konczak, Thomas Linke, and Torsten Schaub. Graphs and colorings for answer set programming. *TPLP*, 6(1-2):61–106, 2006. `doi:10.1017/S1471068405002528`.

11 Claire Lefèvre, Christopher Béatrix, Igor Stéphan, and Laurent Garcia. ASPeRiX, a first-order forward chaining approach for answer set computing. *TPLP*, 17(3):266–310, 2017.

12 Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala: Updated for Scala 2.12*. Artima Incorporation, USA, 3rd edition, 2016.

13 T. Son and E. Pontelli. Planning for biochemical pathways: A case study of answer set planning in large planning problem instances. In Marina De Vos and Torsten Schaub, editors, *Proceedings of the First International SEA'07 Workshop, Tempe, Arizona, USA*, volume 281 of *CEUR Workshop Proceedings*, pages 116–130, January 2007.

14 The Apache Software Foundation. Apache Hadoop, Spark, and Graphx (websites), 2018. [last accessed Feb. 2018] `http://hadoop.apache.org/`, `https://spark.apache.org/`, `https://spark.apache.org/docs/latest/graphx-programming-guide.html`.