Towards Distributed Computation of Answer Sets

Marco De Bortoli, Federico Igne, Fabio Tardivo, Pietro Totis,
Agostino Dovier and Enrico Pontelli

Dept DMIF, University of Udine, Udine, Italy
Dept CS, New Mexico State University, Las Cruces, NM

June 21 2019, CILC

Answer Set Programming

- The ASP (Answer Set Programming) language is a logic sublanguage born in 2000 for Knowledge Representation, it is based on the stable model semantics;
- Two-phases solving procedure: *grounding* and *solving*;
- *grounding* phase can generate huge programs, which exceeds the resources amount of a single machine.

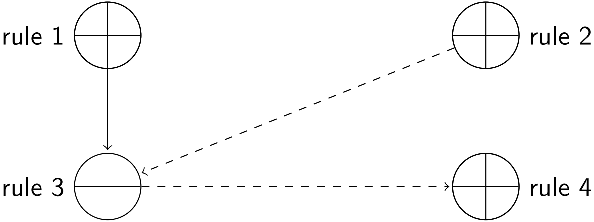
To overcome these limitations, this paper proposes three distributed tools to handle the *grounding* and/or the *solving* phase in a system with distributed resources:

- mASPreduce;
- STRASP (STRAtified ASP);
- DASC (Distributed Answer Set Coloring).

The Graph Coloring Algorithm

Rule Dependancy Graph (RDG)

```
1  p(a).
2  q(b).
3  r(a) :- p(a), not q(b).
4  r(c) :- not r(a).
```



The Graph Coloring Algorithm

Operators

Definition (Nondeterministic operator)

Given the RDG Γ and the partial coloring $\mathcal{C}$ of Γ, $\circ \in \{\oplus, \ominus\}$, we define $\mathcal{D}_\Gamma^\circ : \mathbb{C} \to \mathbb{C}$ as

- $\mathcal{D}_\Gamma^\oplus(\mathcal{C}) = (\mathcal{C}_\oplus \cup \{r\}, \mathcal{C}_\ominus)$ for some $r \in S(\Gamma, \mathcal{C}) \backslash (\mathcal{C}_\oplus \cup C_\ominus)$
- $\mathcal{D}_\Gamma^\ominus(\mathcal{C}) = (\mathcal{C}_\oplus, \mathcal{C}_\ominus \cup \{r\})$ for some $r \in S(\Gamma, \mathcal{C}) \backslash (\mathcal{C}_\oplus \cup C_\ominus)$

Definition (Propagation operators)

$$\mathcal{P}_\Gamma(\mathcal{C}) = \mathcal{C} \sqcup (S(\Gamma, \mathcal{C}) \cap \overline{B}(\Gamma, \mathcal{C}), \overline{S}(\Gamma, \mathcal{C}) \cup B(\Gamma, \mathcal{C}))$$

$$\mathcal{T}_\Gamma(\mathcal{C}) = (\mathcal{C}_\oplus \cup (S(\Gamma, \mathcal{C}) \backslash \mathcal{C}_\ominus), \mathcal{C}_\ominus)$$

$$\mathcal{V}_\Gamma(\mathcal{C}) = (\mathcal{C}_\oplus, \Pi \backslash \mathcal{T}_\Gamma^*(C))$$

The Graph Coloring Algorithm

Solving procedure

Theorem (Operational answer set characterization)

Let Γ RDG and $\mathcal{C}$ total coloring of Γ.
$\mathcal{C}$ is an admissible coloring of Γ iff there exists a sequence $(\mathcal{C}^i)_{0\leq i\leq n}$ such that:

- $\mathcal{C}^0 = (\mathcal{PV})^*_\Gamma((\emptyset,\emptyset))$;
- $\mathcal{C}^{i+1} = (\mathcal{PV})^*_\Gamma(\mathcal{D}^\circ_\Gamma(\mathcal{C}^i))$ for some $\circ \in \{\oplus,\ominus\}$ and $0 \leq i < n$;
- $\mathcal{C}^n = C$.

From $\mathcal{C}^n$ a stable model is deduced.

the mASPreduce solver

- developed by Federico Igne;
- it deals with the pure solving phase only, it is an implementation of the Graph Coloring Algorithm;
- based on MapReduce paradigm;
- written in Scala with the Apache Spark framework;
- RDG encoded via the GraphX module of Spark;
- non-deterministic and propagation operations implemented as map/reduce routines;
- fix point operators implemented with Pregel.

STRASP

- developed by Pietro Totis;
- written in Scala with the Apache Spark framework;
- Dependency graph encoded via the GraphX module of Spark;
- this tool can be used as
  - a grounder for non-definite programs, which speed up the total computation by calculating the maximal stratified subprograms;
  - a complete solver for definitive/stratified programs;
- two levels of parallelization during grounding:
  - Component level parallelism;
  - Rule level parallelism.

DASC

- developed by Marco De Bortoli;
- it deals with the pure solving phase;
- low level implementation of the Graph Coloring Algorithm;
- written in C++ with the Boost library:
  - RDG implemented via Parallel Boost Graph Library;
  - communication implemented via MPI library;
- custom redistribution algorithm;
- design choices w.r.t. mASPreduce:
  - modified version of RDG, in which we also have a node for each atom;
  - different strategy for propagation implementation, which improve network traffic.

Experimental results

DASC

inst	Distr	1 cp unit	2 cp units	3 cp units	4 cp units	5 cp units
2	RR	0.003	0.013	0.015	0.015	0.017
	RD	NR	0.010	0.011	0.013	0.014
3	RR	0.048	0.14	0.19	0.16	0.19
	RD	NR	0.184	0.151	0.166	0.172
4	RR	0.36	1.11	1.42	1.36	1.33
	RD	NR	1.226	1.393	1.115	1.232
5	RR	1.83	6.23	6.18	6.26	5.98
	RD	NR	6.118	6.401	6.226	5.256
6	RR	7.03	22.84	23.86	33.60	24.21
	RD	NR	22.513	20.765	20.881	18.511
7	RR	21.99	71.09	81.55	69.76	65.83
	RD	NR	71.07	83.60	66.43	68.20
8	RR	58.90	188.85	185.45	212.69	220.73
	RD	NR	185.46	195.41	182.33	191.27

Experimental results

mASPreduce

inst	1 cp unit	2 cp units	3 cp units	4 cp units	5 cp units
2	56.330 (0.003)	42.190 (0.010)	40.160 (0.011)	41.177 (0.013)	35.405 (0.014)
3	95.697 (0.048)	64.315 (0.14)	61.767 (0.151)	62.845 (0.16)	54.144 (0.172)
4	150.82 (0.36)	88.043 (1.11)	89.145 (1.393)	89.695 (1.115)	78.178 (1.232)
5	error (1.83)	error (6.118)	error (6.18)	error (6.226)	error (5.256)
6	stopped (7.03)	stopped (22.513)	stopped (20.765)	stopped (20.881)	stopped (18.511)
7	stopped (21.99)	stopped (71.07)	stopped (81.55)	stopped (66.43)	stopped (65.83)
8	stopped (58.90)	stopped (185.46)	stopped (185.45)	stopped (182.33)	stopped (191.27)

Experimental results

STRASP

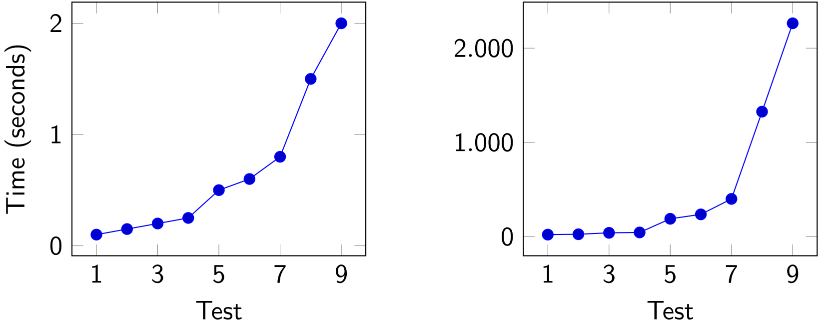


Figure: Comparison between Clingo (left) and our Spark approach to stratified programs (right)

Conclusion and Future Work

- DASC performance shows that lowering the level of implementation pays off, yet we are still far from the state-of-the-art Clingo performance: heuristics implementation would help in that sense;
- Clingo and STRASP have similar trends, diverging only by a constant factor; unfortunately this constant is too large. Anyway, we expected far better results by lowering the implementation level and by adding the Single Rule Level Parallelism.

Thanks for your attention