Verifying resource requirements for distributed rule-based systems

Natasha Alechina, Brian Logan, Nguyen Hoang Nga, and Abdur Rakib*

University of Nottingham, Nottingham, UK
(nza,bsl,hnn,rza)@cs.nott.ac.uk

Abstract. Rule-based systems are rapidly becoming an important component of 'mainstream' computing technologies, for example in business process modelling, the semantic web, sensor networks etc. However, while rules provide a flexible way of implementing such systems, the resulting system behaviour and the resources required to realise it can be difficult to predict. In this paper we consider the verification of system behaviour and *resource requirements* for distributed rule-based systems. More specifically, we consider distributed problem-solving in systems of communicating rule-based systems, and ask how much time (measured as the number of rule firings) and message exchanges does it take the system to find a solution. We show how standard model-checking technology can be used to verify resource requirements for such systems, and present preliminary results which highlight complex tradeoffs between time and communication bounds.

1 Introduction

Rule-based approaches offer significant advantages to the application developer: their focus on the declarative representation of small, relatively independent, knowledge units makes it easier for developers and even end users to rapidly develop and maintain applications — in many cases the information required to develop the application is already codified in terms of rules expressed in natural language, e.g., describing a business process.

However, while the adoption of rule-based approaches brings great benefits in terms of rapid development and ease of maintenance, they also present new challenges to application developers, namely how to ensure the *correctness* of rule-based designs (will a rule-based system produce the correct output for all legal inputs), *termination* (will a rule-based system produce an output at all) and *response time* (how much computation will a rule-based system have to do before it generates an output).

These problems become even more challenging in the case of *distributed rule-based systems*, where the system being designed or analysed consists of several communicating rule-based programs which exchange information via messages, e.g., a semantic web application or a sensor network. A communicated fact (or sensor reading) may be added asynchronously to the state of a RBS while the system is running, potentially

* This work was supported by the Engineering and Physical Sciences Research Council [grant number EP/E031226].

triggering a new strand of computation which executes in parallel with current processing. To be able to provide response time guarantees for such systems, it is important to know how long each rule-based system's reasoning is going to take. In other situations, for example a rule-based system running on a PDA or other mobile device, the number of messages exchanged may be a critical factor.

Verifying properties such as correctness, termination and resource requirements of rule-based systems is extremely challenging. Ironically, the very features which make rule-based systems attractive from a development point of view—the separation of the application logic and execution engine and the ease with which rules can be added or modified (often by end users)—make it hard to predict the overall behaviour of the system or the implications of changing a particular rule. In this paper, we present a framework for the automated verification of time and communication requirements in distributed rule-based systems. We consider distributed problem-solving in systems of communicating rule-based systems, and ask how much time (measured as the number of rule firings) and message exchanges does it take the system to find a solution. We show how standard model-checking technology can be used to solve such problems. Using simple examples, we show how the Mocha model checker [1] can be used to analyse trade-offs between time and communication bounds in a distributed rule-based system.

The structure of the paper is as follows. In section 2 we introduce a simple model of the kinds of distributed rule-based system we want to verify. We describe the encoding of such systems in the input language of Mocha model-checker in section 3. Model-checking experiments are described in section 4. We discuss related work in 5 and conclude in section 6.

2 Distributed rule-based systems

In this section, we introduce a model of a distributed rule-based system and the measures of time and communication resources required to solve a distributed reasoning problem.

We assume that the system consists of n individual rule-based systems or nodes, where $n \geq 1$. Each node is identified by a value in $\{1, \ldots, n\}$, and we use variables i and j over $\{1, \ldots, n\}$ to refer to nodes. Each node i has a *program*, consisting of propositional Horn clause rules, and a working memory, which contains facts (propositions). The restriction to propositional rules is not critical: if the rules do not contain functional symbols and we can assume a fixed finite set of constant symbols, then any set of first-order Horn clauses and facts can be encoded as propositional formulas. If a node i has a rule $A_1, \ldots, A_n \rightarrow B$, the facts $A_1, \ldots, A_n$ are in i's working memory and B is not in i's working memory in state s, then i can fire the rule, adding B to i's working memory in the successor state s'.

In addition to firing rules, nodes can exchange messages regarding facts currently in their working memory. The exchange of information between nodes is modelled as an abstract *Copy* operation: if a fact A is in node i's working memory in state s and A is not in the working memory of node j, then in the successor state s', A can be added to node j's working memory. Intuitively, this corresponds to the following operations

Time	Node 1	Node 2
t_0	$\{A_1, A_2, A_3, A_4\}$	$\{A_5, A_6, A_7, A_8\}$
operation:	RuleB2	RuleB4
t_1	$\{A_1, A_2, A_3, A_4, B_2\}$	$\{A_5, A_6, A_7, A_8, B_4\}$
operation:	RuleB1	RuleB3
t_2	$\{A_1, A_2, A_3, A_4, B_1, B_2\}$	$\{A_5, A_6, A_7, A_8, B_3, B_4\}$
operation:	RuleC1	RuleC2
t_3	$\{A_1, A_2, A_3, A_4, B_1, B_2, C_1\}$	$\{A_5, A_6, A_7, A_8, B_3, B_4, C_2\}$
operation:	Idle	Copy (C_1 from node 1)
t_4	$\{A_1, A_2, A_3, A_4, B_1, B_2, C_1\}$	$\{A_5, A_6, A_7, A_8, B_3, B_4, C_1, C_2\}$
operation:	Idle	RuleD1
t_5	$\{A_1, A_2, A_3, A_4, B_1, B_2, C_1\}$	$\{A_5, A_6, A_7, A_8, B_3, B_4, C_1, C_2, D_1\}$

Fig. 1. Example 1

rolled into one: j asking i for A, and i sending A to j. We assume copy operations are guaranteed to succeed and take one tick of system time. A node can also perform an Idle operation (do nothing).

A problem is considered to be solved if one of the nodes has derived the goal. The time taken to solve the problem is taken to be the total number of steps by the whole system (nodes firing their rules or copying facts in parallel, at most one operation executed by each node at every step). This abstracts away from the cost of rule matching etc. This assumption is made for simplicity and a single 'tick' can be replaced with a numerical value reflecting real time taken by the system to fire a rule (worst case or average). The amount of communication required to solve the problem is taken to be the total number of copy operations performed by all nodes. Note that the only node which incurs the communication cost is the node which performs the copy. As with our model of time, the assumptions regarding communication are made for simplicity; it is straightforward to modify the definition of communication so that, e.g., the 'cost' of communication is paid by both nodes, communication takes more than one tick of time, and communication is non-deterministic.

The execution of a distributed rule-based system can be modelled as a state transition system where states correspond to combined states of nodes (set of facts in each node's working memory) and transitions correspond to nodes performing actions in parallel, where each node's action is either a single rule firing, a copy action, or an idle action.

As an example, consider a system of two nodes, 1 and 2. The nodes share the same set of rules:

$$\textbf{RuleB1}\ A_1, A_2 \rightarrow B_1 \qquad \textbf{RuleB2}\ A_3, A_4 \rightarrow B_2$$

$$\textbf{RuleB3}\ A_5, A_6 \rightarrow B_3 \qquad \textbf{RuleB4}\ A_7, A_8 \rightarrow B_4$$

$$\textbf{RuleC1}\ B_1, B_2 \rightarrow C_1 \qquad \textbf{RuleC2}\ B_3, B_4 \rightarrow C_2$$

$$\textbf{RuleD1}\ C_1, C_2 \rightarrow D_1$$

The goal is to derive D_1. Figure 1 gives a simple example of a run of the system starting from a state where node 1 has A_1, A_2, A_3 and A_4 in its working memory, and node 2 has A_5, A_6, A_7, A_8. In this example, the nodes require one copy operation and five time

steps to derive the goal. (In fact, this is an optimal use of resources for this problem, as verified using model-checking, see section 4).

Throughout the paper, we will use variations on this synthetic 'binary tree' problem, in which the A_is are the leaves and the goal is the root of the tree, as examples. We vary the number of rules and the distribution of 'leaf' facts between the nodes. For example, a larger system can be generated using 16 'leaf' facts $A_1, \ldots, A_{16}$, adding extra rules to derive B_5 from A_9 and A_{10}, etc., and a new goal E_1 derivable from D_1 and D_2. We will refer to this as a '16 leaf example'. We have chosen this sample problem because it is typical of a class of distributed reasoning problems and can be easily parameterised by the number of leaf facts and the distribution of facts and rules among the nodes.

3 Model-checking resource requirements

We are interested in verifying properties of the form 'if the facts $A_1, \ldots, A_n$ are assigned to the nodes of a distributed rule-based system in a particular way, the system will (or will not) conclude Q in less than t timesteps and fewer than m messages'. In general it is impractical to run the system and count steps and messages for all possible interactions between the nodes to establish such properties. What is required is some automated method of verifying such properties which considers all possible system traces.

In this section, we show how the transition system representing a distributed rule-based system can be encoded as an input to a model-checker to allow the automatic verification of the properties expressing resource bounds. Model checking is an automated verification procedure in which the system to be verified is represented by a (finite) model M for an appropriate logic, the property to be verified is represented by a formula ϕ in the same logic, and the verification consists in computing whether M satisfies ϕ [2]. Originally developed for hardware verification, it is increasingly being applied to the verification of complex software systems. For the experiments reported here, we have used the Mocha model checker [1], due to the ease with which we can specify a system of concurrently executing communicating rule-based systems in *reactive modules*, the description language used by Mocha.

In Mocha, the state of the system is described by a set of *state variables* and each system state corresponds to an assignment of values to the variables. The presence or absence of each fact in the working memory of a node is represented by a boolean state variable $n_i A_j$ which encodes node i's belief in fact A_j. The initial values of these variables determines the initial distribution of facts between nodes. [1] In the experiments reported below (which used the binary tree example introduced in the previous section, all derived (non-leaf) variables were initialised to *false*, and only the allocation of leaves to each node was varied.

The actions of firing a rule, copying a fact from another node and idling are encoded as a Mocha *atom* which describes the initial condition and transition relation for a group

[1] We can also leave the initial allocation of facts undetermined, and allow the model checker to find an allocation which satisfies some property, e.g., that there is a derivation which takes less than k steps. However for the experiments reported here, we specified the initial assignment of facts to nodes.

of related state variables. Inference is implemented by marking the consequent of a rule as present in working memory at the next cycle if all of the antecedents of the rule are present in working memory at the current cycle. A rule is only enabled if its consequent is not already present in working memory at the current cycle. Communication is implemented by copying the value representing the presence of a fact in the working memory of another node at the current cycle to the corresponding state variable in the node performing the copy at the next cycle and incrementing a counter, $n_i\_counter$, for the node performing the copy. Copying is only enabled if the fact to be copied is not already in the working memory of the node performing the copy. In the experiments, we assumed that all rules are believed by all nodes in the initial state, and did not implement copying rules. However, this can be done in a straightforward way by adding an extra boolean variable to the premises of each rule, and implementing copying a rule as copying this variable. To express the communication bound, we use a counter for each node which is incremented each time a copy action is performed by the node. To allow a node to idle at any cycle, the atoms which update working memory in each node are declared to be *lazy*.

Mocha supports hierarchical modelling through composition of *modules*. A module is a collection of atoms and a specification of which of the state variables updated by those atoms are visible from outside the module. In our encoding, each node is represented by a module. A particular distributed rule-based system is then simply a parallel composition of the appropriate node modules.

The evolution of the system's state is described by an initial round followed by an infinite sequence of update rounds. The variables are initialised to their initial values in the initial round and new values are assigned to the variables in the subsequent update rounds. At each update round, Mocha non-deterministically chooses between the enabled rules and copy operations, and idling for each node.

The specification language of Mocha is ATL. We can express properties such as 'node i may derive fact ϕ in k steps' as $EX^k\alpha$, where EX^k is EX repeated k times, and α is a state variable encoding of the fact that ϕ is present in node i's working memory (e.g. : $\alpha = n_iA_j$ if $\phi = A_j$). [2] To bound the number of messages used, we can include a bound on the value of the message counter of one or more nodes in the property to be verified. For example, the property 'node i may derive fact ϕ in k steps using at most one message' can be encoded as $EX^k(\alpha \wedge c)$ where c is a boolean variable which is true if $n_i\_counter < 2$. To obtain the actual derivation, we can verify an invariant which states that α is never true, and use the counterexample trace generated by the model-checker to show how the system reaches the state where α is proved.

4 Experimental results

In this section we describe the results of experiments for different sizes of the binary tree example and different distributions of leaves between the nodes. The experiments

[2] In [3] we showed that, given a distributed reasoning system with m nodes, p propositional variables, r propositional rules, and t the largest upper bound on the inference transition in any node, the problem of whether such a temporal property ϕ is true in the system is decidable in time $O(|\phi| \times 2^{m(p+r)} \times t^m)$.

were designed to investigate trade-offs between the number of steps and the number of messages exchanged (a shorter derivation with more messages or a longer derivation with fewer messages).

Case	Node 1	Node 2	# steps	# messages node 1	# messages node 2
1.	A_1–A_8		7	–	–
2.	A_1–A_7	A_8	6	0	3
3.	A_1–A_7	A_8	6	1	2
4.	A_1–A_7	A_8	7	1	1
5.	A_1–A_7	A_8	8	1	0
6.	A_1–A_6	A_7,A_8	6	0	2
7.	A_1–A_6	A_7,A_8	6	1	1
8.	A_1–A_6	A_7,A_8	7	1	0
9.	A_1–A_4	A_5–A_8	5	1	0
10.	A_1,A_3,A_5,A_7	A_2,A_4,A_6,A_8	7	2	3
11.	A_1,A_3,A_5,A_7	A_2,A_4,A_6,A_8	11	0	4

Table 1. Resource requirements for optimal derivation in 8 leaves cases

Table 1 shows the number of derivation steps and the number of messages for each node for varying distributions of 8 leaves. Note that there are several optimal (non-dominated) derivations for the same initial distribution of leaves between the nodes. For example, when node 1 has all the leaves apart from A_8, and node 2 has A_8, the obvious solution is case 5, which requires 1 message and 8 time units: node 1 copies A_8 from node 2, and then derives the goal in 7 inference steps. However, the nodes can solve the problem in fewer steps by exchanging more messages. For example, case 2 describes the situation when node 2 copies A_7 from node 1, while node 1 derives B_3 (step 1). Then node 2 derives B_4 while node 1 derives B_2 (step 2). Then node 2 copies B_3 from node 1, while node 1 derives B_1 (step 3). At the next step node 1 derives C_1 and node 2 derives C_2 (step 4). Then node 2 copies C_1 from node 1 (step 5) and node 1 idles; finally at step 6 node 2 derives D_1. This derivation requires 6 time steps and 3 messages. The trade-off between steps and messages varies with the distribution, as can be seen in cases 10 and 11: if node 1 has all the odd leaves and node 2 all the even leaves, then to derive the goal either requires 7 steps and 5 messages, or 11 steps and 4 messages.

Similar trade-offs are apparent for a problem with 16 leaves, as shown in Table 2. However in this case there are a larger number of possible distributions of leaves, and, in general, more trade-offs for each distribution. The trade-offs are also more dramatic, for example in the 'odd and even' case (cases 20 and 21), where node 1 has all the odd leaves and node 2 all the even leaves, increasing the message bound by 1 reduces the length of the derivation by 10 steps.

Although these examples are very simple, they point to the possibility of complex trade-offs between time and communication bounds in distributed rule-based systems. For more complex examples, we would anticipate that such trade-offs would be harder to predict *a priori*, and our framework would be of correspondingly greater utility.

Case	Node 1	Node 2	# steps	# copy 1	# copy 2
1.	A_1–A_{16}		15	–	–
2.	A_1–A_{15}	A_{16}	12	0	6
3.	A_1–A_{15}	A_{16}	12	1	4
4.	A_1–A_{15}	A_{16}	13	1	3
5.	A_1–A_{15}	A_{16}	14	1	2
6.	A_1–A_{15}	A_{16}	15	1	1
7.	A_1–A_{15}	A_{16}	16	1	0
8.	A_1–A_{14}	A_{15},A_{16}	11	0	5
9.	A_1–A_{14}	A_{15},A_{16}	11	1	4
10.	A_1–A_{14}	A_{15},A_{16}	12	1	3
11.	A_1–A_{14}	A_{15},A_{16}	13	1	2
12.	A_1–A_{14}	A_{15},A_{16}	14	1	1
13.	A_1–A_{14}	A_{15},A_{16}	15	1	0
14.	A_1–A_{12}	A_{13},A_{14},A_{15},A_{16}	11	0	4
15.	A_1–A_{12}	A_{13},A_{14},A_{15},A_{16}	11	1	2
16.	A_1–A_{12}	A_{13},A_{14},A_{15},A_{16}	12	1	1
17.	A_1–A_{12}	A_{13},A_{14},A_{15},A_{16}	13	1	0
18.	A_1–A_3,A_5–A_7,A_9–A_{11},A_{13}–A_{15}	A_4,A_8,A_{12},A_{16}	13	2	6
19.	A_1–A_3,A_5–A_7,A_9–A_{11},A_{13}–A_{15}	A_4,A_8,A_{12},A_{16}	19	4	0
20.	A_1,A_3,A_5,A_7,A_9,A_{11},A_{13},A_{15}	A_2,A_4,A_6,A_8,A_{12},A_{14},A_{16}	13	4	5
21.	A_1,A_3,A_5,A_7,A_9,A_{11},A_{13},A_{15}	A_2,A_4,A_6,A_8,A_{12},A_{14},A_{16}	23	0	8

Table 2. Resource requirements for optimal derivation in 16 leaves cases

5 Related work

The upper limit on deliberation (or response) time in rule-based systems is a well-established problem. However previous work has studied expert and diagnostic systems as single isolated systems [4], and has focused mainly on termination (or worst case response time), rather than more general issues of resource bounds, and trade-offs between time and communication resources.

There exists considerable work on the execution properties of rule based systems, both in AI and in the active database community. In AI, perhaps the most relevant work on the execution properties of rule based systems is that of Cheng and co-workers on predicting the response time of OPS5-style production systems. For example, in [5], Chen and Cheng show how to compute the response time of a rule-based program in terms of the maximum number of rule firings and the maximum number of basic comparisons made by the Rete network. In [6], Cheng and Tsai describe a tool for detecting the worst-case response time of an OPS5 program by generating inputs which are guaranteed to force the system into worst-case behaviour, and timing the program with those inputs. However the results obtained using these approaches are specific to a particular rule-based system (OPS5 in this case) and cannot easily be extended to systems with different rule formats or rule execution strategies. Nor are they capable of dealing with the asynchronous inputs found in communicating RBSs.

Another relevant strand of work is the problem of termination and query boundedness in deductive databases [7]. However, again this work considers a special (and

rather restricted with respect to rule format and execution strategy) class of rule-based systems. In our previous work, e.g., [8, 9] we have investigated time vs. memory trade-offs for single (rule-based and resolution) reasoners, and in [10], we investigated resource requirements for time, memory and communication for systems of distributed resolution reasoners. In [3] we showed how durations can be assigned to the various stages of the inference cycle (matching, conflict resolution etc.) and how abstraction techniques can be used to model sets of individual rule firings into a single abstract transition with associated upper and lower time bounds.

6 Conclusions

In this paper, we proposed an approach to modelling and verifying resource requirements of distributed rule-based systems. We described results of experiments on a synthetic example which show interesting trade-offs between time required by the nodes in a distributed rule-based system to solve the problem and the number of messages they need to exchange. The paper presents initial results of a long-term research programme. In future work, we plan to evaluate our approach on real-life examples of rule-based systems.

References

1. Alur, R., Henzinger, T.A., Mang, F.Y.C., Qadeer, S., Rajamani, S.K., Tasiran, S.: MOCHA: Modularity in model checking. In: Proceedings of 10th International Conference on Computer Aided Verification (CAV'98). (1998) 521–525
2. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, (1999)
3. Alechina, N., Logan, B.: Verifying bounds on deliberation time in multi-agent systems. In Proceedings of the Third European Workshop on Multiagent Systems (EUMAS'05). (2005) 25–34
4. Georgeff, M.P., Lansky, A.L.: Reactive reasoning and planning. In: Proceedings of the Sixth National Conference on Artificial Intelligence, AAAI-87. (1987) 677–682
5. Chen, J.R., Cheng, A.M.K.: Predicting the response time of OPS5-style production systems. In: Proceedings of the 11th Conference on Artificial Intelligence for Applications. (1995) 203
6. Cheng, A.M.K., yen Tsai, H.: A graph-based approach for timing analysis and refinement of OPS5 knowledge-based systems. IEEE Transactions on Knowledge and Data Engineering **16** (2004) 271–288
7. Brodsky, A., Sagiv, Y.: On termination of datalog programs. In: International Conference on Deductive and Object-Oriented Databases (DOOD). (1989) 47–64
8. Albore, A., Alechina, N., Bertoli, P., Ghidini, C., Logan, B., Serafini, L.: Model-checking memory requirements of resource-bounded reasoners. In: Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI 2006). (2006) 213–218
9. Alechina, N., Bertoli, P., Ghidini, C., Jago, M., Logan, B., Serafini, L.: Verifying space and time requirements for resource-bounded agents. In Proceedings of the Fourth Workshop on Model Checking and Artificial Intelligence (MoChArt-2006). (2006) 16–30
10. Alechina, N., Logan, B., Nga, N.H., Rakib, A.: Verifying time, memory and communication bounds in systems of reasoning agents. In Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008). (2008) 736–743