Experiments with DECENTMON2

Tom Cornebize and Yliès Falcone
University of Grenoble - Alps

Abstract. This short report describes some of the experiments carried with DECENTMON2.

1 Implementation and Experimental Results

1.1 DECENTMON2: a Benchmark for Generalized Decentralized Monitoring

DECENTMON2 is an implementation that contains a completely redeveloped version of DECENTMON [1] and an implementation of the decentralized monitoring algorithm presented in a recently submitted paper. DECENTMON2 consists of 1,300 LLOC, written in the functional programming language OCaml. It can be freely downloaded and run from [2]. The system takes as input multiple traces (that can be automatically generated), corresponding to the behaviour of a distributed system, and specification given by a deterministic finite-state automaton. Then the specification is monitored against the traces in two different modes: a) by merging the traces to a single, global trace and then using a "central monitor" for the formula (i.e., all local monitors send their respective events to the central monitor who makes the decisions regarding the trace), b) by using the decentralized version introduced in [1], and c) by using the decentralized approach introduced in this paper (i.e., each trace is read by a separate monitor in the last two cases). To favor the centralized case, monitors send their events only if they differ from the previous one, which decreases the number of exchanged messages. We have evaluated the three different monitoring approaches (i.e., centralized vs. LTL-decentralized vs generalized-decentralized) using several set-ups described in the remainder of this section. To compare monitoring metrics obtained with the decentralized algorithm in [1] and the one of this paper, we have used LTL2MON [3], which converts any LTL formula into an automata-based (centralized) monitor. For our comparison purposes, we used results on common LTL formulae and traces using the experimental setup depicted in Fig. 1. For each of the metric mentioned in the following sections, ratios are obtained by dividing the value obtained in the centralized case over the value obtained in the decentralized case.

To compare with the decentralized monitoring algorithm obtained in [1], the emission of events occurs at the same rate as the communication between monitors. Recall that it was assumed in [1] whereas our monitoring algorithm allows different ratios.

Each line of the following arrays is obtained by conducting 1,000 tests, each with a fresh trace of 1,000 events and specification. We used three different architectures:

- a system of 3 components with local alphabets $\{a\}, \{b\}, \{c\}$;
- a system of 3 components with local alphabets $\{a1, a2\}, \{b1, b2\}, \{c1, c2\}$;
- a system of 6 components with local alphabets $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}$.

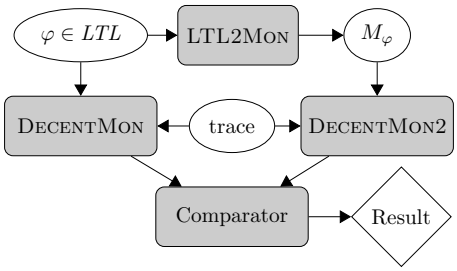

Fig. 1: Experimental setup for comparing DECENTMON and DECENTMON2

In the following, we refer to these architectures as architectures 1, 2, and 3 respectively. In the following monitoring metrics, we measure the size of the elements exchanged by monitors as follows. Suppose we are monitoring a formula φ with set of atomic propositions AP and for which the associated automaton is defined over the alphabet $\Sigma = 2^{AP}$ with set of states Q: each event is of size $\lceil \log_2 |\Sigma| \rceil$, each state is of size $\lceil \log_2 |Q| \rceil$, each time unit t is of size $\lceil \log_2(t) \rceil$, each formula is of size $n \times \lceil \log_2(|AP| + |Op|) \rceil$ where n is the number of symbols in the formula, AP is the set of atomic propositions of the the formula and $Op = \{\top, \bot, \vee, \wedge, \neg, \Rightarrow, \Leftrightarrow, \mathbf{X}, \mathbf{F}, \mathbf{G}, \mathbf{U}, \mathbf{R}, \mathbf{W}, \overline{\mathbf{X}}, \#, (,)\}$ is the set of symbols in formulae handled by DECENTMON. Then in the following tables, the following metrics are used: # msg., the total number of exchanged messages, |msg.|, the total size of exchanged messages (in bits), |trace |the size of the prefix needed to obtain a verdict, delay, the number of additional events needed to reach a verdict compared to the centralized algorithm, |mem|, the memory in bits needed for the structures (i.e., formulae for [1], partial function mem plus state for our algorithm).

1.2 Benchmarks for Randomly Generated Formulae

For each size of formula (from 1 to 6), DECENTMON2 randomly generated 1,000 formulae in the architecture described in Example **??**. How the three monitoring approaches compared on these formulae can be seen in Tables 1a and 2a. The first column shows the size of the monitored LTL formulae. Note, our system measures formula size in terms of the operator entailment[1] inside it (state formulae excluded), e.g., $\mathbf{F}a \wedge \mathbf{G}(b \wedge c)$ is of size 2. The entry |trace| denotes the average length of the traces needed to reach a verdict. For example, the last line in Table 2 says that we monitored 1,000 randomly generated LTL formulae of size 6. On average, monitors using the centralized algorithm, the decentralized algorithm using LTL formulae, and the decentralized algorithm using automata, exchanged 9.09e1, 4.73e1, 1.16e1 messages, had messages of size 2.72e2, 2.84e5, 1.18e3 bits, respectively. The last two pairs of columns show the ratios of the previous metrics obtained in the decentralized cases over the centralized one. For instance, the decentralized algorithm with LTL formulae induced 5.2e-1 of the messages of the centralized one, whereas the decentralized algorithm with automata induced 1.23e0 messages.

1.3 Benchmarks for Patterns of Formulae

We also conducted benchmarks with more realistic specifications, obtained from specification patterns [4]. Actual formulae underlying the patterns are available at [5] and

[1] Experiments show that operator entailment is more representative of how difficult it is to progress it in a decentralized manner. Formulae of size above 6 are not realistic in practice.

Table 1: Number and size of messages for random formulae - Architecture 1

(a) Architecture 1

$\|\varphi\|$	# msg.			|msg.|			# msg. ratio		|msg.| ratio	
1	3.49e0	1.13e0	3.73e0	1.04e1	8.72e1	2.38e1	3.2e-1	1.06e0	8.31e0	2.27e0
2	4.04e0	1.89e0	5.4e0	1.21e1	3.16e2	3.92e1	4.6e-1	1.33e0	2.60e1	3.23e0
3	9.33e0	5.34e0	1.69e1	2.79e1	3.22e3	1.66e2	5.7e-1	1.37e0	1.15e2	4.5e0
4	2.51e1	1.26e1	3.59e1	7.53e1	8.43e3	3.50e2	5.0e-1	1.27e0	1.12e2	4.16e0
5	3.97e1	2.19e1	7.10e1	1.19e2	3.65e4	7.75e2	5.5e-1	1.33e0	3.06e2	4.86e0
6	9.09e1	4.73e1	1.16e2	2.72e2	2.84e5	1.18e3	5.2e-1	1.23e0	1.04e3	4.21e0

(b) Architecture 2

$\|\varphi\|$	# msg.			|msg.|			# msg. ratio		|msg.| ratio	
1	4.25e0	1.48e0	4.66e0	2.55e1	2.26e2	4.87e1	3.4e-1	1.09e0	8.87e0	1.9e0
2	5.16e0	2.38e0	6.54e0	3.10e1	9.26e2	7.68e1	4.6e-1	1.26e0	2.98e1	2.47e0
3	1.38e1	4.88e0	1.93e1	8.33e1	3.49e3	2.39e2	3.5e-1	1.05e0	4.19e1	2.17e0
4	3.15e1	1.25e1	3.76e1	1.89e2	3.73e4	4.59e2	3.9e-1	9.8e-1	1.97e2	2e0
5	7.86e1	2.80e1	8.51e1	4.72e2	1.19e5	9.38e2	3.5e-1	8.6e-1	2.53e2	1.58e0
6	8.44e1	3.23e1	8.51e1	5.06e2	3.23e5	9.51e2	3.8e-1	8.7e-1	6.38e2	1.62e0

(c) Architecture 3

$\|\varphi\|$	# msg.			|msg.|			# msg. ratio		|msg.| ratio	
1	7.82e0	3.93e0	1.39e1	4.69e1	1.01e3	4.63e2	5e-1	1.77e0	2.15e1	9.88e0
2	9.87e0	5.92e0	2.30e1	5.92e1	3.93e3	8.78e2	5.9e-1	2.33e0	6.64e1	1.48e1
3	1.45e1	8.36e0	3.10e1	8.71e1	1.49e4	1.21e3	5.7e-1	2.14e0	1.72e2	1.4e1
4	5.03e1	1.88e1	1.14e2	3.02e2	8.65e4	4.37e3	3.7e-1	1.93e0	2.86e2	1.22e1
5	9.99e1	3.27e1	6.28e2	5.99e2	5.00e5	3.17e4	3.2e-1	5.4e0	8.34e2	4.54e1
6	1.26e2	4.44e1	1.30e2	7.61e2	1.49e6	3.33e3	3.5e-1	9.5e-1	1.95e3	4.07e0

recalled in [2]. To generate formulae, we proceeded as follows. For each pattern, we randomly select one of the formulae associated to it. Such a formula is "parametrized" by some atomic propositions. To obtain randomly generated formula, using the distributed alphabet, we randomly instantiate atomic propositions.

Results are reported in Tables 3 and 4 for each kind of pattern (absence, existence, bounded existence, universal, precedence, response, precedence chain, response chain, constrained chain), we generated again 1,000 formulae, monitored over the same architecture as the ones used previously.

1.4 Conclusions from the Experiments and Discussion

The number and size of exchanged messages when monitoring with the decentralized algorithm using automata are in the same order of magnitude (and most often lower) than when monitoring with the centralized algorithm. Comparing the decentralized monitoring algorithm, the number of messages when using LTL formulae is always lower but the size of messages is much bigger in that case (sometimes by orders of magnitude). Delays are always greater by using automata but remains in the same order of magnitude. Please also note that we have conducted benchmarks where our algo-

Table 2: Trace length, delay, and memory size for random formulae - Architecture 1

(a) Architecture 1

$\|\varphi\|$	|trace |			delay		|mem |	
1	1.33e0	1.66e0	2.61e0	3.2e-1	1.28e0	4.42e1	7.93e0
2	1.67e0	2.15e0	3.2e0	4.8e-1	1.53e0	1.56e2	9.72e0
3	5.21e0	5.79e0	8.8e0	5.8e-1	1.6e0	4.58e2	1.04e1
4	1.57e1	1.64e1	1.93e1	7e-1	1.66e0	1.10e3	1.13e1
5	2.55e1	2.64e1	3.63e1	8.2e-1	1.79e0	2.63e3	1.24e1
6	5.94e1	6.02e1	6.32e1	7.6e-1	1.66e0	5.83e3	1.20e1

(b) Architecture 2

$\|\varphi\|$	|trace |			delay		|mem |	
1	1.57e0	2.01e0	2.86e0	4.4e-1	1.29e0	1.05e2	1.20e1
2	1.97e0	2.6e0	3.44e0	6.3e-1	1.47e0	3.83e2	1.39e1
3	5.82e0	6.59e0	9.46e0	7.7e-1	1.64e0	9.97e2	1.59e1
4	1.36e1	1.44e1	1.83e1	8.3e-1	1.74e0	2.50e3	1.73e1
5	3.46e1	3.54e1	4.53e1	8.3e-1	1.76e0	3.50e3	1.80e1
6	3.71e1	3.79e1	4.47e1	8e-1	1.66e0	6.30e3	1.73e1

(c) Architecture 3

$\|\varphi\|$	|trace |			delay		|mem |	
1	1.62e0	2.42e0	4.58e0	8e-1	2.96e0	3.72e2	3.63e1
2	2.29e0	3.51e0	5.6e0	1.22e0	3.31e0	1.31e3	4.11e1
3	3.84e0	5.36e0	7.37e0	1.52e0	3.55e0	4.06e3	4.52e1
4	1.56e1	1.74e1	2.22e1	1.71e0	3.62e0	9.88e3	4.74e1
5	3.23e1	3.40e1	4.13e1	1.72e0	3.47e0	1.81e4	4.64e1
6	4.12e1	4.29e1	4.78e1	1.63e0	3.41e0	2.84e4	4.56e1

rithm uses only one leader monitor, which tends to augment the delay (whereas in the algorithm using LTL formulae monitors are not constrained) - see the discussion below. Regarding the size of memory, the algorithm using automata is always more efficient by several orders of magnitude when the size of formulae grows.

Efficiency of Implementation. Another interesting feature of our algorithm is its usability in implementation. To illustrate this point, we measured the real memory consumption of the two (reasonably optimized) implementations of benchmarks (in the same programming language). For space reasons, we only report the results below when monitoring one formula of type bounded existence, over alphabet $\{a\}, \{b\}, \{c\}$, with a trace of 10,000 events. For other formulae, the trend is similar. As expected, progression is certainly more costly and less appropriate for monitoring and the size of messages (and hence the size of the formulae) monitors have to handle becomes unmanageable quite rapidly.

Influence of the number of leaders. We also made some experiments (omitted for space reasons) regarding the influence of the number of leader monitors. It turns out that, as the number of leaders augments in the system, the number of messages augments,

	# msg.	|msg.|	|mem |	time
DECENTMON	367	$21,667,225$	$157,845$	$4.724s$
DECENTMON2	$3,258$	$59,628$	18	$0.064s$

whereas the delay induced by decentralized monitoring reduces. For instance, by allowing all monitors to communicate spontaneously (i.e., with $\text{leader\_mon}(i) = true$ for any $i \in [1; n]$), we observed that, for several patterns of formulae, i) a shorter average delay and less memory consumption by a factor of 1.5, and ii) the total size of messages was, in average, multiplied by 1.7 while their number was multiplied by 2 (thus the average size of message decreased).

References

1. Bauer, A.K., Falcone, Y.: Decentralised ltl monitoring. In Giannakopoulou, D., Méry, D., eds.: FM. Volume 7436 of LNCS., Springer (2012) 85–100
2. Cornebize, T., Falcone, Y.: DECENTMON2 (2013) Available at http://decentmon2.forge.imag.fr.
3. Bauer, A.K.: LTL2MON (2009) Available at http://ltl3tools.sourceforge.net.
4. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Intl. Conf. on Software Engineering (ICSE), ACM (1999) 411–420
5. Alavi, H., Avrunin, G., Corbett, J., Dillon, L., Dwyer, M., Pasareanu, C.: Specification patterns website (2011) http://patterns.projects.cis.ksu.edu/.

Table 3: Number and size of messages for specification patterns

(a) Architecture 1

φ	# msg.			|msg.|			# msg. ratio		|msg.| ratio	
abs	7.33e0	4.46e0	1.79e1	2.2e1	2.05e3	1.94e2	6e-1	2.44e0	9.36e1	8.85e0
exis	4.39e1	1.97e1	6.42e1	1.31e2	1.02e4	6.63e2	4.5e-1	1.46e0	7.76e1	5.03e0
bexis	6.53e1	3.16e1	3.79e2	1.96e2	1.17e6	5.45e3	4.8e-1	2.17e0	5.97e3	1.04e1
univ	1.03e1	5.92e0	3.09e1	3.10e1	2.75e3	3.79e2	5.7e-1	2.98e0	8.86e1	1.22e1
prec	7.76e1	2.54e1	6.81e1	2.32e2	8.71e3	6.48e2	3.2e-1	1.29e0	3.74e1	4.11e0
resp	9.59e2	4.25e2	1.07e3	2.87e3	3.37e5	9.76e3	4.4e-1	1.12e0	1.17e2	3.39e0
precc	7.68e0	4.81e0	1.89e1	2.30e1	5.18e3	2.18e2	6.2e-1	2.47e0	2.25e2	9.53e0
respc	6.43e2	3.81e2	7.32e2	1.92e3	7.19e5	6.68e3	5.9e-1	1.13e0	3.72e2	3.46e0
consc	4.90e2	2.01e2	4.69e2	1.47e3	3.37e5	4.26e3	4.1e-1	1.13e0	2.29e2	3.43e0

(b) Architecture 2

Alphabet $\{a1, a2\}, \{b1, b2\}, \{c1, c2\}$										
φ	# msg.			|msg.|			# msg. ratio		|msg.| ratio	
abs	1.93e1	7.12e0	3.46e1	1.16e2	7.18e3	5.66e2	3.6e-1	1.79e0	6.19e1	4.88e0
exis	1.37e2	3.37e1	1.58e2	8.24e2	4.07e4	2.34e3	2.4e-1	1.15e0	4.93e1	2.84e0
bexis	1.96e2	5.64e1	5.77e2	1.18e3	2.42e6	9.87e3	2.8e-1	1.42e0	2.05e3	4.06e0
univ	4.10e1	1.18e1	7.36e1	2.46e2	1.29e4	1.30e3	2.8e-1	1.79e0	5.26e1	5.31e0
prec	3.88e1	1.29e1	4.87e1	2.33e2	1.38e4	7.18e2	3.3e-1	1.33e0	5.92e1	3.26e0
resp	1.47e3	4.66e2	1.20e3	8.86e3	9.12e5	1.26e4	3.1e-1	8.1e-1	1.02e2	1.42e0
precc	2.41e1	8.82e0	4.60e1	1.44e2	3.32e4	8.08e2	3.6e-1	1.9e0	2.29e2	5.57e0
respc	1.19e3	4.64e2	9.20e2	7.15e3	2.80e6	8.74e3	3.8e-1	7.7e-1	3.91e2	1.22e0
consc	7.33e2	2.40e2	4.92e2	4.39e3	8.33e5	4.66e3	3.2e-1	7.7e-1	1.89e2	1.21e0

(c) Architecture 3

Alphabet $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}$										
φ	# msg.			|msg.|			# msg. ratio		|msg.| ratio	
abs	2.33e1	1.22e1	9.69e1	1.40e2	2.33e4	4.61e3	5.2e-1	4.14e0	1.66e2	3.29e1
exis	1.55e2	4.55e1	3.88e2	9.32e2	8.67e4	1.63e4	2.9e-1	2.49e0	9.30e1	1.74e1
bexis	2.71e2	8.01e1	1.47e3	1.62e3	9.91e6	6.60e4	2.9e-1	2.79e0	6.09e3	2.08e1
univ	6.21e1	2.25e1	2.25e2	3.72e2	4.50e4	1.05e4	3.6e-1	3.62e0	1.20e2	2.82e1
prec	5.39e1	1.98e1	1.52e2	3.23e2	4.83e4	6.63e3	3.6e-1	2.98e0	1.49e2	2.17e1
resp	1.98e3	5.64e2	1.83e3	1.18e4	2.32e6	4.36e4	2.8e-1	9.2e-1	1.95e2	3.67e0
precc	3.37e1	1.57e1	1.65e2	2.02e2	1.50e5	7.77e3	4.6e-1	4.9e0	7.41e2	3.84e1
respc	1.65e3	5.31e2	1.60e3	9.93e3	6.55e6	3.78e4	3.2e-1	9.6e-1	6.59e2	3.8e0
consc	9.37e2	2.78e2	6.28e2	5.62e3	2.15e6	1.07e4	2.9e-1	7.5e-1	3.83e2	2.12e0

Table 4: Trace length, delay, and memory size for specification patterns

(a) Architecture 1

φ	|trace |			delay		|mem |	
abs	3.89e0	4.55e0	5.66e0	6.6e-1	1.77e0	4.96e2	1.24e1
exis	2.82e1	2.89e1	2.99e1	6.5e-1	1.68e0	3.76e2	1.17e1
bexis	4.26e1	4.31e1	1.16e2	5.8e-1	1.56e0	2.82e4	1.44e1
univ	5.96e0	6.73e0	7.76e0	7.6e-1	1.79e0	4.98e2	1.30e1
prec	5.08e1	5.16e1	3.55e1	8.1e-1	1.66e0	6.63e2	1.15e1
resp	6.38e2	6.39e2	6.39e2	3.2e-1	7e-1	1.54e3	8.61e0
precc	4.11e0	4.82e0	5.72e0	7e-1	1.64e0	1.20e3	1.16e1
respc	4.27e2	4.28e2	4.28e2	5.9e-1	1.16e0	4.65e3	1.07e1
consc	3.25e2	3.25e2	3.26e2	6e-1	1.35e0	2.72e3	1.08e1

(b) Architecture 2

Alphabet $\{a1, a2\}, \{b1, b2\}, \{c1, c2\}$							
φ	|trace |			delay		|mem |	
abs	8.27e0	9.13e0	1.01e1	8.6e-1	1.83e0	1.10e3	1.95e1
exis	6.06e1	6.15e1	6.23e1	8.3e-1	1.67e0	1.05e3	1.79e1
bexis	8.70e1	8.78e1	1.81e2	8e-1	1.56e0	5.05e4	2.33e1
univ	1.78e1	1.88e1	1.98e1	1.01e0	1.94e0	1.21e3	2.10e1
prec	1.69e1	1.79e1	1.76e1	9.3e-1	1.67e0	1.63e3	1.73e1
resp	6.56e2	6.57e2	6.57e2	4.3e-1	7.1e-1	5.82e3	1.38e1
precc	1.03e1	1.11e1	1.20e1	8.2e-1	1.64e0	4.02e3	1.80e1
respc	5.30e2	5.30e2	5.31e2	5.7e-1	9.5e-1	2.88e4	1.50e1
consc	3.25e2	3.26e2	2.84e2	7.6e-1	1.41e0	9.55e3	1.72e1

(c) Architecture 3

Alphabet $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}$							
φ	|trace |			delay		|mem |	
abs	6.79e0	8.48e0	1.06e1	1.69e0	3.84e0	3.85e3	5.15e1
exis	5.07e1	5.23e1	5.44e1	1.69e0	3.73e0	3.22e3	5.10e1
bexis	8.93e1	9.08e1	1.78e2	1.45e0	3.44e0	2.47e5	6.17e1
univ	1.96e1	2.16e1	2.37e1	2e0	4.13e0	3.87e3	5.66e1
prec	1.69e1	1.88e1	1.96e1	1.97e0	3.77e0	5.96e3	5.01e1
resp	6.59e2	6.60e2	6.61e2	9.3e-1	1.51e0	2.16e4	2.86e1
precc	1.03e1	1.21e1	1.39e1	1.86e0	3.66e0	1.60e4	5.00e1
respc	5.50e2	5.51e2	5.53e2	1.15e0	1.92e0	8.06e4	3.27e1
consc	3.11e2	3.12e2	2.81e2	1.71e0	3.04e0	3.86e4	4.33e1