Formal Verification Problems in a Bigdata World:
Towards a Mighty Synergy

Matteo Camilli
matteo.camilli@unimi.it
http://camilli.di.unimi.it
Outline

• Introduction, Motivations, Objectives
• Background
• Some details on:
  • MapReduce
  • Techniques, Frameworks and Tools
• Experiments
• Conclusion
• Planned work
Introduction

- background on formal methods
  - Modeling
  - Interpreting

- deploy techniques into software tools able to analyze large amount of data very reliably and efficiently
  - adapting an application for exploiting the scalability provided by cloud computing facilities.
Introduction

- background on formal methods
  - Modeling
  - Interpreting

- deploy techniques into software tools able to analyze large amount of data very reliably and efficiently

- adapting an application for exploiting the scalability provided by cloud computing facilities.
Background

• The behavior of a discrete-event dynamic system is formally given in terms of a labeled state transition system: $(S, \Lambda, \rightarrow)$

• $\Lambda$ is a set of labels

• $\rightarrow \subseteq S \times \Lambda \times S$ s.t. $(s, \lambda, s') \in \rightarrow$ iff $s'$ reachable from $s$ (written as $s^\lambda \rightarrow s'$)
Background

- In general $S$ may be infinite, or even uncountable. Some abstraction techniques are required in order to be able to enumerate the whole state space.

- Abstract State Space: $(A, L, \Rightarrow)$

- Where $A$ is a coverage of $S$, and $\Rightarrow \subseteq A \times L \times A$ s.t. exists a morphism $f$ which maps $A$ labels into $L$ labels.
• The relation $\Rightarrow$ satisfies the condition EE:

1. If $a \Rightarrow a'$, then $\exists s \in a, s' \in a'$ : $\lambda \Rightarrow s' \Rightarrow \lambda$ with $\lambda \in f^I(l)$

2. If $s \Rightarrow s'$, then $\forall a \in A$ s.t. $s \in a$, $\exists a' \in A$ s.t. $s' \in a'$ $\land a \Rightarrow a'$
Time Basic nets - Reachability analysis

• Three key points of the Time Reachability Graph building algorithm allow in many cases the termination.
  • Identification of inclusions between classes of states
  • Erasure of absolute times
  • Identification of anonymous timestamps

Time Basic nets - Reachability analysis

• Three key points of the Time Reachability Graph building algorithm allow in many cases the termination.
  • Identification of inclusions between classes of states
  • Erasure of absolute times

Execution of the Gas Burner example:
Total built abstract states: 22,978
Final abstract state space: 14,563

<table>
<thead>
<tr>
<th>architecture</th>
<th># CPUs</th>
<th>tool version</th>
<th>compute model</th>
<th>T</th>
<th>H</th>
<th>f</th>
<th>exec. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4Ghz Intel Core 2 Duo, 2GB RAM</td>
<td>1×2 cores</td>
<td>sequential</td>
<td>local (single machine)</td>
<td>-</td>
<td>-</td>
<td>(2)</td>
<td>~7.5 hrs</td>
</tr>
</tbody>
</table>

Sequential algorithm

Model m

\[ m.\text{buildRoot}() \]

Remaining

State space

\[ f = S_k.\text{getFeatures}() \]

for \( S_j \) in stateSpace.get(f)

\[ S_k.\text{identifyRelationship}(S_j) \]

EQUALS, INCLUDED, INCLUDES, NONE
Sequential algorithm

Model $m$

$m$.buildRoot()

$S_0$

$S_i$

$S_k$

$S_i$.createSuccessors($m$)

State space

Straightforward, but because of the state explosion problem sequential tools may become very slow or even crash.

for $S_j$ in stateSpace.get($f$)

$S_k$.identifyRelationship($S_j$)

EQUALS, INCLUDED, INCLUDES, NONE
Map-Reduce

- Map-Reduce job =
  - **Map** function (inputs -> key-value pairs) +
  - **Reduce** function (key and list of values -> outputs)
- Map and Reduce tasks apply Map and Reduce function to many inputs in parallel.
Map-Reduce TB nets analysis tool

- **Map step =**
  - given an unexplored state, it applies the `createSuccessors` function. **Incoming transitions** are stored into destination states by a list of identifiers.

- **Shuffle step =**
  - Gathers together states potentially related: This is done by using as intermediate keys the evaluation of the `getFeatures` function.

- **Reduce step =**
  - given a set of states potentially related, it applies the `identifyRelationship` function foreach pair of states.

- **Building blocks =**
  - State = `<M,C>` pair. M marking, C constraint.
  - `identifyRelationship` computes the actual relationship between two states according to the following rule: \( a \subseteq a' \iff \sigma(M) = \sigma(M') \land C \Rightarrow C' \)
  - `getFeatures` returns just the topological part of \( M \equiv \sigma(M) \).
Hybrid Iterative Map-Reduce

- A single Map-Reduce job is not enough: Iterative Map-Reduce
- During the first and last iterations of the algorithm the set of states is quite small. Thus a MapReduce job over a large cluster of machines is useless and expensive in term of time and resources.
- The computation starts with a sequential algorithm and goes on until the state space size passes a configurable threshold. After that we distribute the computation over a big cluster.
Hybrid Iterative Map-Reduce

• A single Map-Reduce job is not enough: *Iterative Map-Reduce*

• During the first and last iterations of the algorithm the set of states is quite small. Thus a MapReduce job over a large cluster of machines is useless and expensive in terms of time and resources.

- The computation starts with a sequential algorithm and goes on until the state space size passes a configurable threshold. After that we distribute the computation over a big cluster.

**Gas Burner example:**

<table>
<thead>
<tr>
<th>#machines</th>
<th>machine type</th>
<th>#abstract states</th>
<th>threshold</th>
<th>time (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>m2.2xlarge</td>
<td>1.456x10</td>
<td>200</td>
<td>175</td>
</tr>
<tr>
<td>4</td>
<td>m2.2xlarge</td>
<td>1.456x10</td>
<td>200</td>
<td>95</td>
</tr>
<tr>
<td>8</td>
<td>m2.2xlarge</td>
<td>1.456x10</td>
<td>200</td>
<td>39</td>
</tr>
</tbody>
</table>

• The execution with 8 machines is almost 80% faster than the sequential algorithm we distribute the computation over a big cluster.
MapReduce-based Distributed building of reachability GraphS
Use Cases

• P/T nets
  • State = $<M>$ marking, associates places with natural numbers.
  • $s = s' \iff M = M'$ thus we can use the optimized Reduce phase.

• In order to prove the effectiveness of using MaRDiGraS to improve legacy tools, we adapted an existing P/T nets tool: PIPE.

• To adapt the sequential algorithm of PIPE into a distributed one, we just needed 290 lines of code: a very small number also if compared with the dimension of the effectively used PIPE modules (~6500 lines of code).
Use Cases

Shared Memory example:
- \(1.831 \times 10^6\) reachable states
- The PIPE tool takes more than 20 hours to complete the computation.
- The adapted version takes 74 min to complete the same computation, using 16 machines.

Simple Load Balancing example:
- \(4.060 \times 10^8\) states
- \(3.051 \times 10^9\) transitions
- 120GB of data
- Execution time = 530 min. using 20 machines.
CTL model checking in the cloud

• We developed a software tool which exploits the MaRDiGraS computed graphs by applying iterative map-reduce algorithms based on fixpoint characterizations of the basic temporal operators of CTL (Computational Tree Logic).

• Given a state transition system $T = \langle S, s_0, R, L \rangle$, and a set of states that satisfy the $\phi$ formula ($[\phi]_T$)
  
  - $[\text{EX}\phi]_T = R^-([\phi]_T)$
  - $[\text{EG}\phi]_T = \nu_X([\phi]_T \cap R^-(X))$
  - $[\text{E}[\phi U \psi]]_T = \mu_X([\psi]_T \cup ([\phi]_T \cap R^-(X)))$
Computation Tree Logic

• CTL is a branching-time logic which models time as a tree-like structure where each moment can be followed by several different possible futures. In CTL each basic temporal operator (i.e., either $X$, $F$, $G$) must be immediately preceded by a path quantifier (i.e., either $A$ or $E$). In particular, CTL formulas are inductively defined as follows:

$$
\phi ::= p \mid \neg \phi \mid \phi \lor \psi \mid A \psi \mid E \psi \text{ (state formulas)}
$$

$$
\psi ::= X \phi \mid F \phi \mid G \phi \mid \phi U \psi \text{ (path formulas)}
$$

• The interpretation of a CTL formula is defined over a Kripke structure (i.e, a state transition system).

Definition 1 (Kripke structure): A Kripke structure $T$ is a quadruple $\langle S, S_0, R, L \rangle$, where:

1) $S$ is a finite set of states.
2) $S_0$ is the set of initial states.
3) $R \subseteq S \times S$ is a a total transition relation, that is: $\forall s \in S \exists s' \in S$ such that $(s, s') \in R$
4) $L : S \rightarrow 2^{AP}$ labels each state with the set of atomic propositions that hold in that state.
It can be shown that any CTL formula can be written in terms of $\neg$, $\lor$, $\text{EX}$, $\text{EG}$, and $\text{EU}$.

\[ R^-(W) := \{ s \in S : \exists s'(R(s, s') \land s' \in W) \} \]

\[ [\text{EX}\phi]_T = R^-([[\phi]_T]) \]

\[ [\text{EG}\phi]_T = \nu_X([[\phi]_T \cap R^-(X))] \]

\[ [\text{E}[\phi U \psi]]_T = \mu_X([\psi]_T \cup ([[\phi]_T \cap R^-(X))]) \]
MapReduce EX evaluation

\[ [EX\phi]_T = R^-(\phi)_T \]

**Algorithm 2** MapReduce algorithm for evaluating EXφ

1: function MAP(k, s)
2:     if \( s \in \phi_T \) then
3:         for \( e \in R^-(s) \) do
4:             emit(e, ⊥)
5:         end for
6:     end if
7:     emit(k, s)
8: end function
9: function REDUCE(k, list := [s₁, s₂, ...])
10:    if \( ⊥ \in list \) then
11:        s := s' ∈ list s.t. s' ≠ ⊥
12:        emit(k, s)
13:    end if
14: end function
MapReduce EG evaluation

\[ [EG\phi]_T = \nu_X ([\phi]_T \cap R^-(X)) \]

Algorithm 3 MapReduce for evaluating $EG\phi$

1: function MAP$(k, s)$
2:     if $s \in X$ then
3:         for $e \in R^-(s)$ do
4:             emit$(e, \bot)$
5:         end for
6:     end if
7:     if $s \in [\phi]_T$ then
8:         emit$(k, s)$
9:     end if
10: end function
11: function REDUCE$(k, list := [s_1, s_2, \ldots])$
12:     if $\bot \in list \land (s \neq \bot \in list)$ then
13:         emit$(k, s)$
14:     end if
15: end function
\[ E[\phi U \psi]_T = \mu_X ([\psi]_T \cup ([\phi]_T \cap R^{-}(X))) \]

**Algorithm 4** MapReduce algorithm for evaluating \( E[\phi U \psi] \)

1: \textbf{function} MAP\((k, s)\)
2: \hspace{1em} if \( s \in X \) then
3: \hspace{2em} for \( e \in R^{-}(s) \) do
4: \hspace{3em} emit\((e, \bot)\)
5: \hspace{2em} end for
6: \hspace{1em} end if
7: \hspace{1em} if \( s \in [\phi]_T \lor s \in [\psi]_T \) then
8: \hspace{2em} emit\((k, s)\)
9: \hspace{1em} end if
10: \textbf{end function}
11: \textbf{function} REDUCE\((k, \text{list} := [s_1, s_2, \ldots])\)
12: \hspace{1em} \text{list} := list \text{ s.t. } s \neq \bot
13: \hspace{1em} if \((\bot \in \text{list} \land s \neq \text{null}) \lor (s \in [\psi]_T)\) then
14: \hspace{2em} emit\((k, s)\)
15: \hspace{1em} end if
16: \textbf{end function}
CTL experiments

• Models:
  • Shared memory (~10^6 states, ~10^7 transitions)
  • Dekker (~10^7 states, ~10^8 transitions)
  • Simple load balancing (~10^8 states, ~10^9 transitions)

Table 1: Shared memory report

<table>
<thead>
<tr>
<th>property</th>
<th>cardinality</th>
<th># machines</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EX φ</td>
<td>2.135 × 10^6</td>
<td>1</td>
<td>70</td>
</tr>
<tr>
<td>EX φ</td>
<td>2.135 × 10^6</td>
<td>2</td>
<td>67</td>
</tr>
<tr>
<td>EX φ</td>
<td>2.135 × 10^6</td>
<td>4</td>
<td>50</td>
</tr>
<tr>
<td>EX φ</td>
<td>2.135 × 10^6</td>
<td>8</td>
<td>38</td>
</tr>
<tr>
<td>EG ψ</td>
<td>0</td>
<td>1</td>
<td>67</td>
</tr>
<tr>
<td>EG ψ</td>
<td>0</td>
<td>2</td>
<td>55</td>
</tr>
<tr>
<td>EG ψ</td>
<td>0</td>
<td>4</td>
<td>58</td>
</tr>
<tr>
<td>E[ω U ρ]</td>
<td>1.831 × 10^6</td>
<td>1</td>
<td>1898</td>
</tr>
<tr>
<td>E[ω U ρ]</td>
<td>1.831 × 10^6</td>
<td>2</td>
<td>1124</td>
</tr>
<tr>
<td>E[ω U ρ]</td>
<td>1.831 × 10^6</td>
<td>4</td>
<td>839</td>
</tr>
<tr>
<td>E[ω U ρ]</td>
<td>1.831 × 10^6</td>
<td>8</td>
<td>564</td>
</tr>
<tr>
<td>E[ω U ρ]</td>
<td>1.831 × 10^6</td>
<td>16</td>
<td>509</td>
</tr>
</tbody>
</table>

Table 2: Dekker report

<table>
<thead>
<tr>
<th>property</th>
<th>cardinality</th>
<th># machines</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EX φ</td>
<td>1.153 × 10^7</td>
<td>1</td>
<td>660</td>
</tr>
<tr>
<td>EX φ</td>
<td>1.153 × 10^7</td>
<td>2</td>
<td>532</td>
</tr>
<tr>
<td>EX φ</td>
<td>1.153 × 10^7</td>
<td>4</td>
<td>241</td>
</tr>
<tr>
<td>EX φ</td>
<td>1.153 × 10^7</td>
<td>8</td>
<td>144</td>
</tr>
<tr>
<td>EX φ</td>
<td>1.153 × 10^7</td>
<td>16</td>
<td>120</td>
</tr>
<tr>
<td>EG ψ</td>
<td>7.405 × 10^6</td>
<td>1</td>
<td>1567</td>
</tr>
<tr>
<td>EG ψ</td>
<td>7.405 × 10^6</td>
<td>2</td>
<td>1356</td>
</tr>
<tr>
<td>EG ψ</td>
<td>7.405 × 10^6</td>
<td>4</td>
<td>517</td>
</tr>
<tr>
<td>EG ψ</td>
<td>7.405 × 10^6</td>
<td>8</td>
<td>391</td>
</tr>
<tr>
<td>EG ψ</td>
<td>7.405 × 10^6</td>
<td>16</td>
<td>287</td>
</tr>
<tr>
<td>E[ω U ρ]</td>
<td>5.767 × 10^6</td>
<td>1</td>
<td>1357</td>
</tr>
<tr>
<td>E[ω U ρ]</td>
<td>5.767 × 10^6</td>
<td>2</td>
<td>1063</td>
</tr>
<tr>
<td>E[ω U ρ]</td>
<td>5.767 × 10^6</td>
<td>4</td>
<td>585</td>
</tr>
<tr>
<td>E[ω U ρ]</td>
<td>5.767 × 10^6</td>
<td>8</td>
<td>454</td>
</tr>
<tr>
<td>E[ω U ρ]</td>
<td>5.767 × 10^6</td>
<td>16</td>
<td>372</td>
</tr>
</tbody>
</table>

Table 3: Simple load balancing report

<table>
<thead>
<tr>
<th>property</th>
<th>cardinality</th>
<th># machines</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EX φ</td>
<td>1.716 × 10^8</td>
<td>1</td>
<td>2908</td>
</tr>
<tr>
<td>EX φ</td>
<td>1.716 × 10^8</td>
<td>2</td>
<td>2401</td>
</tr>
<tr>
<td>EX φ</td>
<td>1.716 × 10^8</td>
<td>4</td>
<td>937</td>
</tr>
<tr>
<td>EX φ</td>
<td>1.716 × 10^8</td>
<td>8</td>
<td>693</td>
</tr>
<tr>
<td>EX φ</td>
<td>1.716 × 10^8</td>
<td>16</td>
<td>251</td>
</tr>
<tr>
<td>EG ψ</td>
<td>4.060 × 10^8</td>
<td>1</td>
<td>21678</td>
</tr>
<tr>
<td>EG ψ</td>
<td>4.060 × 10^8</td>
<td>2</td>
<td>17147</td>
</tr>
<tr>
<td>EG ψ</td>
<td>4.060 × 10^8</td>
<td>4</td>
<td>6525</td>
</tr>
<tr>
<td>EG ψ</td>
<td>4.060 × 10^8</td>
<td>8</td>
<td>2983</td>
</tr>
<tr>
<td>EG ψ</td>
<td>4.060 × 10^8</td>
<td>16</td>
<td>1226</td>
</tr>
<tr>
<td>E[ω U ρ]</td>
<td>7.524 × 10^7</td>
<td>1</td>
<td>1821</td>
</tr>
<tr>
<td>E[ω U ρ]</td>
<td>7.524 × 10^7</td>
<td>2</td>
<td>1714</td>
</tr>
<tr>
<td>E[ω U ρ]</td>
<td>7.524 × 10^7</td>
<td>4</td>
<td>602</td>
</tr>
<tr>
<td>E[ω U ρ]</td>
<td>7.524 × 10^7</td>
<td>8</td>
<td>377</td>
</tr>
<tr>
<td>E[ω U ρ]</td>
<td>7.524 × 10^7</td>
<td>16</td>
<td>203</td>
</tr>
</tbody>
</table>
CTL experiments

(a) Dekker model checking time
(b) Dekker Speedup
(c) Dekker efficiency

(d) Simple-lb model checking time
(e) Simple-lb Speedup
(f) Simple-lb efficiency
Conclusion

• **MaRDiGraS + CTL verification in the cloud** allow users to implement distributed reachability graph builders and verification tools for different formalisms without care about all non functional aspects.
  
  • They apply techniques typically used by the big data community and so far poorly explored for this kind of issues.

• We believe that this work could be a first step towards a synergy between two very different, but related communities: the formal verification community and the big data community.

• **Open Questions**
  
  • How it can be optimized when the remaining set gets very small?
  
  • How to choose the optimal threshold dynamically?
  
  • Are there classes of formalisms for which this approach cannot be used? And how can we adapt it to these classes?
  
  • ... ?
Planned Work

• Development of a technique for tackling topologically infinite TB net models
  • computation of minimal coverability sets (so far unexplored)
  • this provides a means to decide several important properties also for real time systems:
    • coverability: is it possible to reach a marking dominating a given marking?
    • boundedness: is the set of reachability markings finite?
    • place boundedness: is it possible to bound the number of tokens in a given place?
    • semi-liveness: is there a reachable marking in which a given transition is enabled?
References


