A Declarative Approach to Distributed Computing: Specification, Execution & Analysis

Jiefei Ma, Franck Le, David Wood
Alessandra Russo, Jorge Lobo

Imperial College, IBM, ICREA-Univeristat Pompeu Fabra
Path-vector Routing Table

<table>
<thead>
<tr>
<th>Destination</th>
<th>SPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>[b]</td>
</tr>
<tr>
<td>c</td>
<td>[c]</td>
</tr>
<tr>
<td>d</td>
<td>[c,d]</td>
</tr>
</tbody>
</table>
Distance-vector Routing Table

<table>
<thead>
<tr>
<th>Destination</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>d</td>
<td>b</td>
</tr>
</tbody>
</table>
Path-vector Routing Table

<table>
<thead>
<tr>
<th>Destination</th>
<th>SPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>[b]</td>
</tr>
<tr>
<td>c</td>
<td>[c]</td>
</tr>
<tr>
<td>d</td>
<td>[c,d]</td>
</tr>
</tbody>
</table>

\[
\text{path}(S, D, [S, D], 1) : - \ link(S, D).
\text{path}(S, D, [S|P], C+1) : - \\
\quad \text{link}(S, Z), \ \text{path}(Z, D, C), \\
\quad S =/= D, \ S \ not \ in \ P.
\text{bestCost}(S, D, \min\{C\}) : - \\
\quad \text{path}(S, D, P, C).
\text{routeTable}(S, D, SPath) : - \\
\quad \text{bestCost}(S, D, C), \\
\quad \text{path}(S, D, SPath, C).
\]
Declarative Networking

<table>
<thead>
<tr>
<th>Destination</th>
<th>SPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>[b]</td>
</tr>
<tr>
<td>c</td>
<td>[c]</td>
</tr>
<tr>
<td>d</td>
<td>[c,d]</td>
</tr>
</tbody>
</table>

\[
\text{path}(\text{@S}, \text{D}, [\text{S}, \text{D}], 1) :\text{ link}(\text{@S}, \text{D}).
\]

\[
\text{path}(\text{@S}, \text{D}, [\text{S}\mid \text{P}], \text{C}+1) :\text{ link}(\text{@S}, \text{Z}), \text{path}(\text{@Z}, \text{D}, \text{C}), \\
\hspace{1cm} \text{S} = \neq \text{D}, \text{ S not in P.}
\]
Declarative Networking

<table>
<thead>
<tr>
<th>Destination</th>
<th>SPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>[b]</td>
</tr>
<tr>
<td>c</td>
<td>[c]</td>
</tr>
<tr>
<td>d</td>
<td>[c,d]</td>
</tr>
</tbody>
</table>

path(@S,D,[S,D],1) :- link(@S,D).
path(@S,D,[S|P],C+1) :-
  link(@S,Z), path(@Z,D,C),
  S=/=D, S not in P.

Copying the program in each node, doing a bottom-up computation and placing the ground atoms in the right location, the network eventually computes all the paths.
Declarative Networking

<table>
<thead>
<tr>
<th>Destination</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>[b]</td>
</tr>
<tr>
<td>c</td>
<td>[c]</td>
</tr>
<tr>
<td>d</td>
<td>[c,d]</td>
</tr>
</tbody>
</table>

linkr(@Z,S) :- link(@S,Z).
path(@S,D,[S,D],1) :- link(@S,D).
path(@S,D,[S|P],C+1) :-
    linkr(@Z,S), path(@Z,D,C),
    S=/=D, S not in P.

By re-writing the rules, implementations may decide where rule bodies are evaluated
Declarative Networking

The distributed evaluation of these rules is more complicated - How long should a node wait before it can compute the aggregation?

<table>
<thead>
<tr>
<th>Destination</th>
<th>SPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>[b]</td>
</tr>
<tr>
<td>c</td>
<td>[c]</td>
</tr>
<tr>
<td>d</td>
<td>[c,d]</td>
</tr>
</tbody>
</table>

\[
\text{bestCost}(\$S, D, \text{min}\langle C \rangle) :-
\text{path}(\$S, D, P, C).
\]

\[
\text{routeTable}(\$S, D, P) :-
\text{bestCost}(\$S, D, C),
\text{path}(\$S, D, P, C).
\]
Declarative Networking

And, in general, changes of state needed to be captured when there are changes in the topology are captured outside logic.
Our Approach: Distributed State Machine
An I/O automaton defined using a dialect of C+ in which

• States: are sets of fluents
• Messages and Application Input: are two disjoint sets of actions called communication and input actions respectively

And F is defined using static, dynamic and communication laws
A Voting Algorithm Example

- Each node has an initial opinion of either good (blue) or bad (red)
- Nodes communicate their opinions to their neighbors
- Node switches opinion if the majority of nodes in the neighborhood are of the opposite opinion
- And informs of the change to the neighbors
- Repeat
Voting DSM\(i\) Schema

\textbf{caused} neighbour\_opinion(X, O) \textbf{after} send\_vote(O, X, i).

\textbf{caused} neighbour\_opinion(X, OldO) \textbf{after}

\hspace{1em} neighbour\_opinion(X, OldO), not send\_vote(AnyO, X, i).

\textbf{caused} num\_bad( \#count\ltk X\rtk ) \textbf{if} neighbour\_opinion(X, bad).

\textbf{caused} num\_good( \#count\ltk X\rtk ) \textbf{if} neighbour\_colour(X, good).

\textbf{caused} my\_opinion( bad) \textbf{if}

\hspace{1em} num\_bad( B), num\_good( G), B \geq G.

\textbf{caused} my\_opinion( good) \textbf{if}

\hspace{1em} num\_bad( B), num\_good( G), B < G.

\textbf{sent} send\_vote(O, i, X) \textbf{if}

\hspace{1em} my\_opinion(O), neighbour(X),

\hspace{1em} \textbf{after} my\_opinion( OldO), O \neq OldO.
Connecting DSMs

- **I/O automaton composition**: a composition is realized by identifying actions in different automata with the same name as the same action, and compositions are allowed if the automata are **compatible**.

- A collection of automata is **compatible** iff all the set of output actions are pair-wise disjoint.
Voting DSM(i) Schema

caused neighbour_opinion(X, O) after receive_vote(O, X, i).
caused neighbour_opinion(X, OldO) after neighbour_opinion(X, OldO), not receive_vote(AnyO, X, i).

caused num_bad( #count<X> ) if neighbour_opinion(X, bad).
caused num_good( #count<X> ) if neighbour_colour(X, good).

caused my_opinion( bad) if
    num_bad( B), num_good( G), B >= G.
caused my_opinion( good) if
    num_bad( B), num_good( G), B < G.

sent send_vote(O, i, X) if
    my_opinion(O) , neighbour(X),
    after my_opinion( OldO), O != OldO.
Connecting DSMs

Encoding of synchronous reliable communication:
\[\text{sent receive\_vote}(V,X,S) \text{ after send\_vote}(V,S,X).\]

Encoding of unreliable communication:
\[\text{inertial values(Val).} \]
\[\text{initially values(succeeds), values(fails).} \]
\[\text{caused comm(#chooce<Val>) if values(Val).} \]
\[\text{sent receive\_vote}(V,X,i) \text{ if comm(succeeds) after send\_vote}(V,i,X).} \]
Specification, Execution and Framework

Declarative Protocol Specification

Setup environment, e.g., DSM

Deployment, Execution

Flawed specification results in slow, non-functional or insecure network

Communication Model

Protocol Model

Simple and direct translation between declarative specification and Datalog+Time (representing the semantics of the protocol)

Convergent? Devoid of forwarding loops? Discover multi-paths?

Initial Network Configuration

Synchronous or Asynchronous? Reliable? Fair?

Logic Program

Query answers as logic models, in which execution traces can be extracted.

Theorem Prover

Network topology? (Configuration/Security) policies?

Queries
Synchronous Sample Statistics

For Voting Example

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Graphs</th>
<th>Convergent Configs.</th>
<th>Divergent Configs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>94</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>661</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>112</td>
<td>7042</td>
<td>126</td>
</tr>
<tr>
<td>7</td>
<td>853</td>
<td>107443</td>
<td>1741</td>
</tr>
<tr>
<td>8</td>
<td>11117</td>
<td>2813729</td>
<td>32223</td>
</tr>
</tbody>
</table>
Convergence in BGP Protocol

Objectives: BGP has been shown to always converge in the absence of dispute wheel. However, in the presence of dispute wheels, a BGP configuration may converge or diverge.

Experiment:
- We implement a simplified version of BGP that considers only the local preference, path length and next hop attributes.
- We adopt the asynchronous communication model.
- We check well-known BGP gadgets with dispute wheels, and increase the dispute wheel size or network size.

Result:

<table>
<thead>
<tr>
<th>Gadget</th>
<th>Total Nodes</th>
<th>Converge?</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gadget A, wheel size 2</td>
<td>3</td>
<td>Sometimes</td>
<td>0.079s</td>
</tr>
<tr>
<td>Gadget A, wheel size 3</td>
<td>4</td>
<td>Never</td>
<td>3.173s</td>
</tr>
<tr>
<td>Gadget A, wheel size 4</td>
<td>5</td>
<td>Sometimes</td>
<td>25.784s</td>
</tr>
<tr>
<td>Gadget A, wheel size 5</td>
<td>6</td>
<td>Never</td>
<td>4m</td>
</tr>
<tr>
<td>Gadget A, wheel size 6</td>
<td>7</td>
<td>Sometimes</td>
<td>102m</td>
</tr>
<tr>
<td>Gadget A, wheel size 7</td>
<td>8</td>
<td>Never</td>
<td>4503m</td>
</tr>
<tr>
<td>Gadget B</td>
<td>5</td>
<td>Always</td>
<td>123m</td>
</tr>
<tr>
<td>Gadget C</td>
<td>11</td>
<td>Sometimes</td>
<td>503m</td>
</tr>
</tbody>
</table>
Final Remarks

- There are many distributed algorithms that can be implemented under this computational model besides routing protocols.
- We would like to scale the analysis at least one order of magnitude.
- We have not done analysis with topology changes ...
- We need better axiomatizations for fairness.
- We don’t have an axiomatization of “open” broadcast.
- ...

The Team

Alessandra Russo
Imperial College

Jiefei Ma
Imperial College

David Wood
IBM

Franck Le
IBM
The End