Fault-Tolerant

Model Checking A Self-Stabilizing Synchronization Protocol For Arbitrary Digraphs

Mahyar R. Malekpour
http://shemesh.larc.nasa.gov/people/mrm/

DASC 2012, October 14 – 18
Outline

• Synchronization

• Verification via formal methods

• Fault spectrum and complexity

• Where are we now and where are we going?
What Is Synchronization?

• Local oscillators/hardware clocks operate at slightly different rates, thus, they drift apart over time.
• Local logical clocks, i.e., timers/counters, may start at different initial values.
• The synchronization problem is to adjust the values of the local logical clocks so that nodes achieve synchronization and remain synchronized despite the drift of their local oscillators.

• Application – Wherever there is a distributed system
• How can we synchronize a distributed system?
• Under what conditions is it (im)possible?
It all started with SPIDER, 1999
(Scalable Processor-Independent Design for Extended Reliability)

- Safety critical systems must deal with the presence of various faults, including arbitrary (Byzantine) faults

- **Goals (in the presence and absence of faults):**
  1. Initialization from arbitrary state
  2. Recovery from random, independent, transient failures
  3. Recovery from massive correlated failures
Why Is Synchronization Problem Difficult?

- Design of a fault-tolerant distributed real-time algorithm is extraordinarily hard and error-prone
  - Concurrent processes
  - Size and shape (topology) of the network
  - Interleaving concurrent events, timing, duration
  - Fault manifestation, timing, duration
  - Arbitrary state, initialization, system-wide upset

- It is notoriously difficult to design a formally verifiable solution for self-stabilizing distributed synchronization problem.
Characteristics Of A Desired Solution

- Self-stabilizes in the presence of various failure scenarios.
  - From any initial random state
  - Tolerates bursts of random, independent, transient failures
  - Recovers from massive correlated failures
- Convergence
  - Deterministic
  - Bounded
  - Fast
- Low overhead
- Scalable
- No central clock or externally generated pulse used
- Does not require global diagnosis
  - Relies on local independent diagnosis
- A solution for $K = 3F+1$, if possible, otherwise, $K = 3F+1+X$, $(X = ?) \geq 0$
and,

must show the solution is correct.
Formal Verification Methods

• Formal method techniques: **model checking**, **theorem proving**

• Use a model checker to verify a possible solution insuring that there are no false positives and false negatives.
  – It is deceptively simple and subject to abstractions and simplifications made in the verification process.

• Use a theorem prover to prove that the protocol is correct.
  – It requires a paper-and-pencil proof, at least a sketch of it.
Model Checking

• Model checking issues
  – State space explosion problem
  – Tools require in-depth and inside knowledge, interfaces are not mature yet
  – Modeling a real-time system using a discrete event-based tool

• Intuitive solution is more memory and more computing power
  – PC with 4GB of memory running Linux, 32bit
  – There is a hardware limitation on the amount of memory that can be added to a given system
  – It may not eliminate/resolve state space problem
Alternatively …

- Find a simpler solution
- Reduce the problem complexity by reducing its scope or restricting the assumptions
- Wait for a more powerful model checker
  - 64-bit tool utilizing more memory
  - Faster and more efficient model checking algorithm
The Big Picture

• Solve the problem in the absence of faults.

• Learn and revisit faulty scenarios later on.
Fault Spectrum

Simple fault classification:
1. None
2. Symmetric
3. Asymmetric (Byzantine)

The OTH (Omissive Transmissive Hybrid) fault model classification based on Node Type and Link Type outputs:

1. Correct (None)
2. Omissive Symmetric
3. Transmissive Symmetric (Symmetric)
4. Strictly Omissive Asymmetric
5. Single-Data Omissive Asymmetric
6. Transmissive Asymmetric (Byzantine)
What About Topology?

- What should the graph look like?
  - Graphs of interest: single ring, double ring, grid, bi-partite, etc.
  - Possible options (Sloane numbers/sequence):

<table>
<thead>
<tr>
<th>$K$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of 1-connected graphs</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>21</td>
<td>112</td>
<td>853</td>
<td>11117</td>
</tr>
</tbody>
</table>

- Example, for 4 nodes there are 6 different graphs:
<table>
<thead>
<tr>
<th>n</th>
<th>a(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>112</td>
</tr>
<tr>
<td>7</td>
<td>853</td>
</tr>
<tr>
<td>8</td>
<td>11117</td>
</tr>
<tr>
<td>9</td>
<td>261080</td>
</tr>
<tr>
<td>10</td>
<td>11716571</td>
</tr>
<tr>
<td>11</td>
<td>1006700565</td>
</tr>
<tr>
<td>12</td>
<td>164059830476</td>
</tr>
<tr>
<td>13</td>
<td>50335907869219</td>
</tr>
<tr>
<td>14</td>
<td>29003487462848061</td>
</tr>
<tr>
<td>15</td>
<td>31397381142761241960</td>
</tr>
<tr>
<td>16</td>
<td>63969560113225176176277</td>
</tr>
<tr>
<td>17</td>
<td>245871831682084026519528568</td>
</tr>
<tr>
<td>18</td>
<td>1787331725248899088890200576580</td>
</tr>
<tr>
<td>19</td>
<td>24636021429399867655322650759681644</td>
</tr>
</tbody>
</table>
Synchronization

• What are the parameters?
  – Maximum number of faults, \( F \geq 0 \)
  – Communication delay, \( D \geq 1 \) clock ticks
  – Network imprecision, \( d \geq 0 \) clock ticks
    • So, communication delay is bounded by \([D, D+d]\)
  – Oscillator drift, \( 0 \leq \rho < 1 \),
  – Number of nodes, i.e., network size, \( K \geq 1 \)
  – Synchronization period, \( P \)
  – Topology, \( T \)

• Synchronization, \( S = (F, D, d, \rho, K, P, T) \)

Realizable Systems

Scalability
Where Are We Now?

- Have a family of solutions for detectably bad faults and $K \geq 1$ that applies to realizable systems.
  - Network impression and oscillator drift

- Have model checked a set of digraphs, NASA/TM-2011-217152
  - As much as our resources allowed (mainly, memory constrained)
  - Sample SMV codes are available at: http://shemesh.larc.nasa.gov/people/mrm/publication.htm

- Have a deductive proof, NASA/TM-2011-217184
  - Concise and elegant
The Protocol

<table>
<thead>
<tr>
<th>Synchronizer:</th>
<th>Monitor:</th>
</tr>
</thead>
</table>
| E0: if \(\text{LocalTimer} < 0\)  
  \(\text{LocalTimer} := 0,\) | case (message from the corresponding node) |
| E1: elseif \(\text{ValidSync}()\) and \(\text{LocalTimer} < D\)  
  \(\text{LocalTimer} := γ,\) \hspace{1em} // interrupted | {Sync:  
  ValidateMessage()} \hspace{1em} Other:  
  Do nothing. \hspace{1em} } // case |
| E2: elseif \((\text{ValidSync}()\) and \(\text{LocalTimer} \geq T_S\))  
  \(\text{LocalTimer} := γ,\) \hspace{1em} // interrupted  
  Transmit Sync, | |
| E3: elseif \(\text{LocalTimer} \geq P\) \hspace{1em} // timed out  
  \(\text{LocalTimer} := 0,\)  
  Transmit Sync, | |
| E4: else  
  \(\text{LocalTimer} := \text{LocalTimer} + 1.\) | |

Mahyar Malekpour, DASC 2012
How Does It Work?

1. If someone is out there – accept its **Sync** message and relay it to others,

2. If no one is out there (or they are too slow) – take charge and generate a new **Sync** message,

3. Ignore – reject all **Sync** messages while in the **Ignore Window**.
   - Rules 1 and 2 result in an endless cycle of transmitting messages back and forth
   - The **Ignore Window** properly stops this endless cycle
Key Results

Global Lemmas And Theorems

How do we know when and if the system is stabilized?

• **Theorem Convergence** – For all $t \geq C$, the network converges to a state where the guaranteed network precision is $\pi$, i.e., $\Delta_{Net}(t) \leq \pi$.

• **Theorem Closure** – For all $t \geq C$, a synchronized network where all nodes have converged to $\Delta_{Net}(t) \leq \pi$, shall remain within the synchronization precision $\pi$.

• **Lemma ConvergenceTime** – For $\rho \geq 0$, the convergence time is $C = C_{Init} + \lceil \Delta_{Init}/\gamma \rceil P$.

• **Theorem Liveness** – For all $t \geq C$, LocalTimer of every node sequentially takes on at least all integer values in $[\gamma, P-\pi]$.
Key Results

Local Theorem

How does a node know when and if the system is stabilized?

- **Theorem Congruence** – *For all nodes* $N_i$ *and for all* $t \geq C$, *(Ni.LocalTimer(t) = γ) implies $ΔNet(t) \leq π.*

Key Aspects Of Our Deductive Proof

1. Independent of topology
2. Realizable systems, i.e., $d \geq 0$ and $0 \leq ρ << 1$
3. Continuous time

Mahyar Malekpour, DASC 2012
Model Checking Propositions

- **SystemLiveness**
  \[ AF(ElapsedTime) \]

- **ConvergenceAndClosure**
  
  \[ AF(ElapsedTime) \land AG((ElapsedTime \land AllWithinPrecision) \rightarrow AX(ElapsedTime \land AllWithinPrecision)) \]  
  - Determinism Property
  
  \[ AG(ElapsedTime \rightarrow AllWithinPrecision) \land AG((ElapsedTime \land AllWithinPrecision) \rightarrow AX(ElapsedTime \land AllWithinPrecision)) \]  
  - Convergence Property
  
  \[ AG((ElapsedTime \land AllWithinPrecision) \rightarrow AX(ElapsedTime \land AllWithinPrecision)) \]  
  - Closure Property

- **Congruence**

  \[ AF(ElapsedTime) \land AG((ElapsedTime \land (Node_1.LocalTimer = g)) \rightarrow AX(ElapsedTime \land AllWithinPrecision)) \]
Model Checking Propositions (cont.)

- **ProtocolLiveness**
  
  \[
  AF (ElapsedTime) \land \\
  AG (((ElapsedTime) \land (Node_1.LocalTimer = i)) \rightarrow \\
  \quad AX ((Node_1.LocalTimer = i) \lor (Node_1.LocalTimer = i+1))) \land \\
  AG (((ElapsedTime) \land (Node_1.LocalTimer = P)) \rightarrow \\
  \quad AX (Node_1.LocalTimer = 0))
  \]

  For all \( i = g .. (P - \pi) \)
## Model Checked Cases

<table>
<thead>
<tr>
<th>$K$</th>
<th><strong>Topology</strong> (all links bidirectional)</th>
<th><strong>Topology</strong> (digraphs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1 of 1</td>
<td>1 of 1</td>
</tr>
<tr>
<td>3</td>
<td>2 of 2</td>
<td>5 of 5</td>
</tr>
<tr>
<td>4</td>
<td>6 of 6</td>
<td>83 of 83</td>
</tr>
<tr>
<td>5</td>
<td>21 of 21</td>
<td>Single Directed Ring</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 Variations of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Doubly Connected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Directed Ring</td>
</tr>
<tr>
<td>6</td>
<td>112 of 112</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>Linear*</td>
<td>Linear*</td>
</tr>
<tr>
<td>7</td>
<td>Star*</td>
<td>Star*</td>
</tr>
<tr>
<td>7</td>
<td>Fully Connected*</td>
<td>Fully Connected*</td>
</tr>
<tr>
<td>7 (3×4)</td>
<td>Fully Connected Bipartite*</td>
<td>Fully Connected Bipartite*</td>
</tr>
<tr>
<td>7</td>
<td>Combo</td>
<td>4 of 4</td>
</tr>
<tr>
<td>7</td>
<td>Grid</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>Full Grid</td>
<td>-</td>
</tr>
<tr>
<td>9 (3×3)</td>
<td>Grid</td>
<td>-</td>
</tr>
<tr>
<td>15</td>
<td>Star*</td>
<td>Star*</td>
</tr>
<tr>
<td>20</td>
<td>Star*</td>
<td>Star*</td>
</tr>
</tbody>
</table>
More Results, In Retrospect

• Our family of solutions handles more than the no-fault (correct) case. It handles cases 1, 2, and 4 of the OTH fault classification. I.e., it is a fault-tolerant protocol as long as our assumptions are not violated and the faulty behavior does not violate our definition of digraph.

• In retrospect, “fault-tolerant” should be included in the paper’s title.

• Our family of solutions is an emergent system.

The OTH (Omissive Transmissive Hybrid) fault model classification based on Node Type and Link Type outputs:

1. Correct (None, No-fault)
2. Omissive Symmetric (Fail-detected, Fail-silent)
3. Transmissive Symmetric (Symmetric)
4. Strictly Omissive Asymmetric (1 or 2)
5. Single-Data Omissive Asymmetric
6. Transmissive Asymmetric (Byzantine)
Questions?