Fault-Tolerant

A Self-Stabilizing Synchronization Protocol For Arbitrary Digraphs

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

PRDC 2011, December 12 – 14
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?
A Brief History of Synchronization

- Norbert Wiener, mathematician
  - Author of the 1950 book *Cybernetics: The Control and Communication in the Animal and the Machine*
  - Brain waves, alpha rhythm, 1954
- Art Winfree, majored in engineering physics, wanted to be biologist
  - Modeled using runners on a track, synchronization in time and space, 1964
  - Topology was a ring
- Yoshiki Kuramoto, physicist
  - Introduced order parameter, synchronization in time, 1975
  - Topology was a ring
- Edsger W. Dijkstra, computer scientist
  - Presented an algorithm for a ring
A Brief History of Synchronization

• Charlie Peskin, applied mathematician
  – Proposed self-organization idea (278 pg), in 1973-1975, while working on cardiac pacemakers.
  – Conjectured that there is a solution
  – Started to prove N-body systems of oscillators for large N
  – Ended with proof for two pulse-coupled oscillators by restricting the problem to its bare bone

• Steven Strogatz and Rennie Mirollo, mathematicians
  – Develop proof for N-pulse-coupled oscillators, 1989
  – Approach was simulation followed by mathematical proof for
    • Ideal case,
    • Ideal oscillators, and
    • Fully connected graph
  – Many publications, including a book entitled SYNC
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
What is known?

- Agreement can be guaranteed only if $K \geq 3F + 1$,
  - $K$ is the total number of nodes and $F$ is the maximum number of Byzantine faulty nodes.
  - E.g. need at least 4 nodes just to tolerate 1 fault.
- Periodic re-synchronization to prevent too much deviation in clocks/timers due to drift.
- There are many partial solutions based on strong assumptions (e.g., initial synchrony, or existence of a common pulse).
- There are clock synchronization algorithms that are based on randomization and are non-deterministic.
- There are claims that cannot be substantiated.
- There are no guidelines for how to solve this problem or documented pitfalls to avoid in the process.
- Speculation on proof of impossibility.
- There is no solution for the general case.
Why is this 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.
The Idea

• Keys: It is a feedback control system.
  It is a non-linear system.

• Bring all good nodes from any state to a known state.
  – *Convergence* property

• Maintain bounded synchrony.
  – *Closure* property
The interplay of *Coarsely* and *Finely Synchronized* protocols.

Any State

Coarse Synchronization

Precision too large?

Fine Synchronization

Mahyar Malekpour, PRDC 2011
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.
Bridging Two Worlds

• From simulation (VHDL) to model checking (SMV, SMART, UPPAL, NuSMV)

• From an engineer to a formal methods practitioner
  – I became a believer and an advocate; a formal methodist

• Found a partial solution in 2003, published in 2006
• Found another partial solution in 2007, published in 2009
• These solutions are for 4 nodes with one Byzantine fault and do not scale well to larger number of Byzantine faults
• Model checking of the first protocol took two years
• Model checking results are publically available
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

• 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
Thus far, we’ve considered only the Byzantine faults and produced partial solutions.

Change In Strategy
– The shortest path between two points is not necessarily a straight line.
– First, 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:
(http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20100028297_2010031030.pdf)
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($T$)?

- In the absence of faults, our previous two protocols work for graphs of any size.
  - Model checked for $K \leq 15$
  - As long as the graph is fully connected

- What about other topologies? What should the graph look like?
  - Other 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>Linear</th>
<th>Star/Hub</th>
<th>-</th>
<th>Ring</th>
<th>-</th>
<th>Complete</th>
</tr>
</thead>
</table>

## Sloane A001349

<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, $D > 0$ clock ticks
  – Network imprecision, $d \geq 0$ clock ticks
    • So, communication delay is bounded by $[D, D+d]$
  – Oscillator drift, $0 \leq \rho \ll 1$,
  – Number of nodes, i.e., network size, $K \geq 1$
  – Synchronization period, $P$
  – Topology, $T$

• Synchronization, $S = (F, D, d, \rho, K, P, T)$
Where Are We Now?

- Have a family of solutions for $F = 0$ and $K \geq 1$ that apply to all of the following scenarios and encompass all of the above parameters.
  1. Ideal scenario where $\rho = 0$ and $d = 0$.
  2. Semi-ideal scenario where $\rho = 0$ and $d \geq 0$.
  3. Non-ideal scenario, i.e., realizable systems, where $\rho \geq 0$ and $d \geq 0$.

- Have model checked a set of digraphs, NASA/TM-2011-217152
  - As much as our resources allowed (mainly, memory constrained)

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

- Other researchers currently model checking graphs with more nodes and fewer abstractions than our model checking effort.

Mahyar Malekpour, PRDC 2011
The Protocol

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

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

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

• **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 \( t \geq C \), \( (N_i.\text{LocalTimer}(t) = \gamma) \) implies \( \Delta \text{Net}(t) \leq \pi \).

Key Aspects Of Our Deductive Proof

1. Independent of topology
2. Realizable systems, i.e., \( d \geq 0 \) and \( 0 \leq \rho << 1 \)
3. Continuous time
## Model Checked Cases

<table>
<thead>
<tr>
<th>$K$</th>
<th>Topology (all links bidirectional)</th>
<th>Topology (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 2 Variations of Doubly Connected 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 (3x4)</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 (3x3)</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>
Variations Of The Protocol

• **It is a family of solutions**

• **Reset**
  in E1 and E2:

  
  
  ```
  LocalTimer := γ, // interrupted
  LocalTimer := 0, // interrupted
  
  ```

• Reduced precision, \( \Delta_{ij} = \gamma \), \( \Delta_{InitGuaranteed} = W\gamma \)

• Ripple effect

• Same usable range \([\gamma, P - \pi]\)
Variations Of The Protocol

- **Jump Ahead**
  in E1 and E2:
  
  ```
  LocalTimer := γ,                           // interrupted
  LocalTimer := LocalTimerIn + γ,          // interrupted
  if (LocalTimer ≥ P)
    LocalTimer := 0,
  
  Transmit Sync,
  Transmit Sync and LocalTimer,
  ```

- Increased overhead (message size)
- More messages during convergence
- Better precision, $\Delta_{Init\text{Guaranteed}} = (1+d)\delta(P)$
- Less usable range $[W_{\gamma}, P - \pi]$
Variations Of The Protocol

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

• The general form, dynamic digraph, $S' = (F, D, d, \rho, K(t), P, T(t))$
  – $K(t)$ represents the dynamic node count at time $t$
  – $T(t)$ represents the dynamic topology for a given $K(t)$

• Dynamic Node Count – the number of nodes comprising the network can change at any time.
• Dynamic Topology – the number of links can change at any time.

• Dynamic Digraph – once synchrony is achieved, the system maintains its synchrony provided that the new nodes enter the network from a reset state.
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)
Status and Issues

- Our deductive proof is documented and publically available. There is still a need for this solution to be analyzed in a more mathematically rigorous way.

- Is there a better way to prove that the protocol is correct?
- How to verify the proofs?
- To model check or to theorem prove?
  - If neither, then what?
  - If model check, then how to model check all topologies?
- Any volunteers?
Where are we going?

- At the end, we are defining the path from one end of the fault spectrum to the other; from No Fault to Byzantine Faults and solving the synchronization problem for the general case, i.e., for all topologies and fault types.

- We envision a final general/unifying solution encompassing all topologies and fault types.

- Note that thus far the convergence time, $C$, seems consistent and, I believe, we are on the right track.

$$C_{Byzantine} = O(P)$$

$$C_{No-fault} = O(P)$$
Questions?