A Self-Stabilizing Hybrid Fault-Tolerant Synchronization Protocol

Mahyar R. Malekpour
NASA-Langley Research Center
mahyar.r.malekpour@nasa.gov
+1 757-864-1513
http://shemesh.larc.nasa.gov/people/mahyar.htm

Background

- Aerospace Operations and Safety Program
- Research on distributed fault-tolerant systems
- Challenges
  - Start up, i.e. initialization
  - Recovery from random, independent, transient failures
  - Recovery from massive correlated failures
  - In other words, must address Self-Stabilization
- Desired features
  - Fast recovery
  - Deterministic solution

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

What is the stabilization of clock synchronization problem?

- In electrical engineering terms, for digital logic and data transfer, a synchronous object requires a clock signal.
- A distributed synchronous system requires a logical clock signal.
- Synchronization means coordination of simultaneous threads or processes to complete a task in order to get correct runtime order and avoid unexpected race conditions.
- Stabilization of clock synchronization is bringing the logical clocks of a distributed system *in sync* with each other.

How to achieve stabilization?

- External Control (centralized, master-target)
  - Direct
    - Power on/Cold Reset
    - Hot Reset
    - Master switch

  $\Bigg\}$ Great for close proximity
  - Indirect
    - GPS, i.e. time (synchronous)
    - Go/Start command (asynchronous)
- Problems
  - GPS is not always available
  - There is no GPS on Mars or the Moon
  - Central command is impractical over long distances

How to achieve synchronization?

- Internal Control (distributed)
  - Local awareness about self and state of the system (diagnosis)
  - Coordination and cooperation with others
- Problems
  - Awareness
  - Establish synchrony/agreement
    - On critical states; schedule, membership
  - Maintain synchrony/agreement

Self-Stabilization

Diagnosis

Convergence

Closure

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 approach

- The approach is dynamic and gradual.
  - It takes time; convergence is not spontaneous
  - Requires continuous vigilance and participation
  - Based on system awareness (feedback), i.e., local diagnosis
  - Understanding the relationship between time and event

- It is a feedback control system.

Analogy – a control system

- Non-linear systems:
  Initial Conditions + Perturbations → Unstable States

- Clock synchronization:
  Initial Conditions + Faulty Behavior → Counterexamples

- **Research topic/idea:**
  – Someone with math and control system background to model and analyze this problem and our solutions.

Is the problem solved yet?

- Not quite.
  - There are solutions for special cases

- Synchronization is still a very active topic in various fields, including:
  - Biology
  - Neurobiology
  - Medicine
  - Sociology
  - Computer Science
  - Engineering
  - Mathematics
  - Geophysics, e.g., Volcanoes

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.
- Re-synchronization cycle or period, P, to prevent too much deviation in clocks/timers.
- There are many partial solutions based on strong assumptions (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.

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, at least faster than existing protocols
- Low overhead
- Scalable
- No central clock or externally generated pulse used
- Does not require global diagnosis
  - Relies on local independent diagnosis
- Find a solution for $3F+1$, if possible, otherwise, $3F+1+X$, $(X = ?) \geq 0$

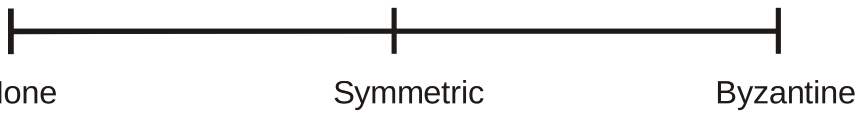
Synchronization parameters

- What are the parameters?
  - Communication delay, $D > 0$ clock ticks
  - Network imprecision, $d \geq 0$ clock ticks
    - So, communication 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
  - Maximum number of faults, $F \geq 0$

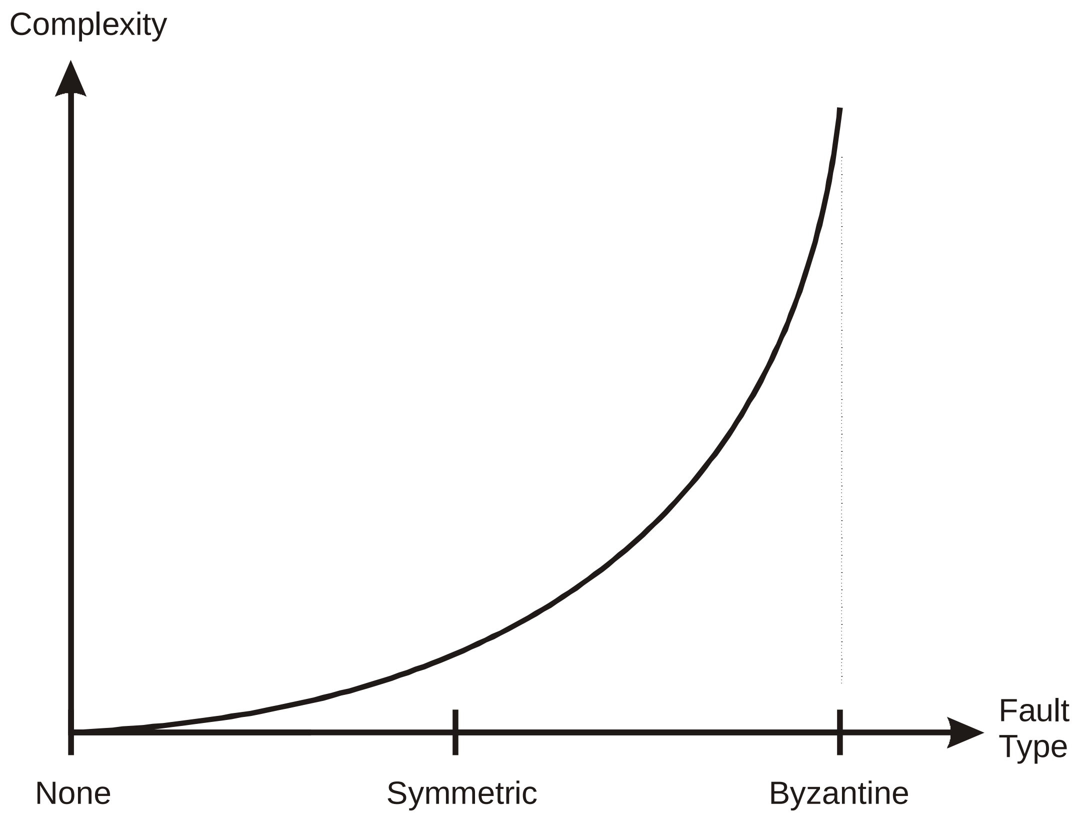
$$\left.\begin{array}{l}D,\ d,\ \rho\end{array}\right\}\ \text{Realizable Systems}\qquad \left.\begin{array}{l}K,\ P,\ T,\ F\end{array}\right\}\ \text{Scalability}$$

- **Synchronization, S = f (K, T, D, d, ρ, P, F)**

Fault spectrum



Fault complexity curve



Where we are

- No (Detectable) Faults
- Symmetric Faults
- Asymmetric Faults

Solutions for detectably bad faults

- No/Detectable Faults (“None” in previous charts)
- Have a family of solutions that apply to all of the following scenarios and encompass all of the above parameters, including arbitrary and dynamic graphs, as long as the definition holds.
  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 paper-and-pencil proofs,
  - Concise and elegant
- Model checked a set of graphs, as many and as varied as our resources (memory, computation) allowed.
- Published in PRDC 2011
- Published in DASC 2012, model checking

Solutions for symmetric faults

- Included in this paper.
- Have a solution that applies to all of the following scenarios, but currently limited to fully connected graphs.
  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 $d \geq 0$.
- Working on a paper-and-pencil proofs for the fully connected graphs.
- Model checked fully connected graphs
  - $F = 1, 2,$ and 3, $D = 1$, $d = 0$, and $\rho \geq 0$
  - $F = 2$ and $D = 1, 2$, $d = 0, 1$, and $\rho \geq 0$
- Generalization to other topologies left for future work.

Solutions for asymmetric faults

- Direct approach
  - I don't believe there is a solution for the general asymmetric (Byzantine) case.
- Indirect approach, two-step process
  1. Convert asymmetry to symmetry
  2. Use a solution for symmetric fault case to solve the problem
- How to convert asymmetry to symmetry?
  1. Using engineering techniques, e.g., pair-wise comparison, lockstep processors, TTTech and their bus guardians is an example, etc.
  2. Oral Message of Lamport *et al.* solves Byzantine Agreement Problem
- Option 1 has good solutions but doesn't guarantee 100% coverage.
- Option 2 provides 100% coverage but is very costly for $F > 2$.
  - Requires $K > 3F$, $2F+1$ disjoint communication paths, $F+1$ rounds of communication, and number of exchanged messages grows exponentially.

Questions?