Software Design for Cyber-Physical Systems

Edward A. Lee

Module 8: Distributed Systems

Technical University of Vienna
Vienna, Austria, May 2022
Example: Google Spanner
A Globally Distributed Database

Distributed database with redundant storage and query handling across data centers.

Update to a record comes in. Time stamp $t$.

Query for the same record comes in. Time stamp $r$. 
Example: Google Spanner
A Globally Distributed Database

Semantics of the database is that it handles queries in timestamp order.

One Possible Approach: Chandy and Misra [1979]

- Assume events arrive reliably in timestamp order.
- Wait for events on each input.
- Process the event with the smaller timestamp.
  - E.g. $r_1 < t_1$
One Possible Approach: Chandy and Misra [1979]

- Deterministic
- Network traffic for “null messages.”
- Every node is a single point of failure.
Another Possible Approach: Jefferson: Time Warp [1985]

- Speculatively execute.
- If a message with an earlier timestamp later arrives...
Another Possible Approach: Jefferson: Time Warp [1985]

- Speculatively execute.
- If a message with an earlier timestamp later arrives...
- Backtrack!
Another Possible Approach: Jefferson: Time Warp [1985]

- No single point of failure.
- Can process events without network traffic
  - Can’t backtrack side effects.
- Overhead: Snapshots
- Uncontrollable latencies.

Platform A:
- Web Server
- Database

Platform B:
- Network Interface
- Database

Diagram showing network traffic between Platform A and Platform B.
A Third Possible Approach: High Level Architecture (HLA)

- Next event request (NER) with $r$
- Next event request (NER) with $t$
- If $r < t$, then time advance grant (TAG) of $q \leq r$
- If $q = r$, process event
• Deterministic.
• RTI is a single point of failure.
• Works well for simulation, but not for online processing.
Ptides/Spanner Approach

- Local clock on each platform.
- \( t \) and \( r \) from local clocks.
- Bounded execution time \( W \).
- Bounded network latency \( L \).
- Event is known at \( B \) by time \( t + W + L \) (by clock at \( A \)).
- Bounded clock synchronization error \( E \).
- Event is known at \( B \) by time \( t + W + L + E \) (by clock at \( B \)).

Event with timestamp \( r \) is safe to process at time \( r + W + L + E \) (by clock at \( B \)).
Ptides/Spanner Approach

- No single point of failure.
- Can process events with no network traffic.
- Latencies are well defined.
- Time thresholds computed statically.
- Assumptions are clearly stated.

When is this “safe to process”? When \( T \geq t + W_1 + E + N \), where

- \( T \) is the local physical clock time
- \( W_1 \) is worst-case execution time
- \( E \) is the bound on the clock synchronization error
- \( N \) the bound on the network delay

[Zhao et al., 2007]
[Edison et al., 2012]
[Corbett et al., 2012]
Using Time Instead of Timeout for Fault-Tolerant Distributed Systems

LESLIE LAMPORT
SRI International

A general method is described for implementing a distributed system with any desired degree of fault-tolerance. Instead of relying upon explicit timeouts, processes execute a simple clock-driven algorithm. Reliable clock synchronization and a solution to the Byzantine Generals Problem are assumed.


General Terms: Design, Reliability

Additional Key Words and Phrases: Clocks, transaction commit, timestamps, interactive consistency, Byzantine Generals Problem

ACM Transactions on Programming Languages and Systems, 1984.

Lee, Berkeley
Abstract: Discrete-event (DE) models are formal system specifications that have analyzable deterministic behaviors. Using a global, consistent notion of time, DE components communicate via time-stamped events. DE models have primarily been used in performance modeling and simulation, where time stamps are a modeling property bearing no relationship to real time during execution of the model. In this paper, we extend DE models with the capability of relating certain events to physical time...
Google independently developed a very similar technique and applied it to distributed databases.
federated reactor {
    c = new Count();
    p = new Print();
    c.out -> p.in;
}

Federated LF Programs
federated reactor at wessel.eecs.berkeley.edu {
    c = new Count();
    p = new Print();
    c.out -> p.in;
}

This will put the RTI (runtime infrastructure) on the specified machine. The federates can go anywhere.
Install the RTI

https://lf-lang.org/docs/handbook/distributed-execution

git clone https://github.com/lf-lang/reactor-c.git
cd reactor-c/core/federated/RTI/
mkdir build && cd build
cmake ../
make
sudo make install

Download Epoch and/or command-line tools from the nightly build (0.2.0 and VS Code extension will not work)
Running by Hand

> RTI -n 2 -i myFedID
RTI: Number of federates: 2
RTI: Federation ID: myFedID
Starting RTI for 2 federates in federation ID myFedID
RTI using TCP port 15045 for federation myFedID.
RTI: Listening for federates.

> bin/Federated_c -i myFedID
Federation ID for executable bin/Federated_c: myFedID
Federate 0: Connected to RTI at localhost:15045.
---- Start execution at time Fri May 13 06:01:30 2022
---- plus 447993000 nanoseconds.
Federate 0: ---- Using 6 workers.
...

> bin/Federated_p -i myFedID
Federation ID for executable bin/Federated_p: myFedID
Federate 1: Connected to RTI at localhost:15045.
---- Start execution at time Fri May 13 06:02:19 2022
---- plus 563334000 nanoseconds.
Federate 1: ---- Using 6 workers.
Federate 1: Starting timestamp is: 1652414540563681000.
Federate 1: ***** Received: 0
Federate 1: ***** Received: 1
Federate 1: ***** Received: 2
...

(0, 1 sec)

out  in
Running Using Script

```bash
> bin/Federated
RTI: Federation ID: 244caf75c3fe2deeda5001d944a256c3637b7c4d796824e5
...
Federate 1: ***** Received: 0
Federate 1: ***** Received: 1
Federate 1: ***** Received: 2
```
Centralized: Enforces deterministic semantics regardless of network delays and execution times (based on HLA). (This is the default.)

Decentralized: Enforces forward progress and detects violations of deterministic semantics when network delays get too large (based on Ptides).
Federated Startup
Initial Connection

Other federates do the same.
Centralized Coordination Only:
Tell RTI of Connection Structure
Clock Synchronization

Synchronize Clocks

Federated

c : Count
(0, 1 sec)

RTI

p : Print

out in
**Precision Time Protocols**

Round-trip delay:

\[ r = (t_4 - t_1) - ((t_3 + e) - (t_2 + e)). \]

where \( e \) is the clock error in the slave. Estimate of the clock error is

\[ \tilde{e} = (t_2 + e) - t_1 - r/2. \]

If communication latency is exactly symmetric, then \( \tilde{e} = e \), the exact clock error. \( B \) calculates \( \tilde{e} \) and adjusts its local clock.
Federated Startup: Determining the Starting Logical Time

The padding $p$ helps physical and logical times to align well at startup.

Physical time $T_1$

Starting logical time $t = \max(T_1, T_2) + p$

Physical time $T_2$

Federated

$c : \text{Count}$

(0, 1 sec)

$p : \text{Print}$

out in
Centralized Coordination: Next Event Request (NET)

RTI

NET(0 sec) No need for a response because there are no upstream federates!

Federated

\[ c : \text{Count} \]

\( (0, 1 \text{ sec}) \)

\[ p : \text{Print} \]
Centralized Coordination: Next Event Request (NET)

No response yet because RTI cannot assure that all messages have been received.

NET (5 secs) (timeout time)

Federated

\( c \): Count

\( (0, 1 \text{ sec}) \)

\( p \): Print

out in
Centralized Coordination: Tagged Message Sending via RTI

- Tagged Message (0 sec, 0) goes through the RTI
- TAG (0 sec, 0) (Tag Advance Grant)

Advance tag to (0 sec, 0) and invoke reaction.
Centralized Coordination: Next Event Request (NET)

RTI

No need for a response because there are no upstream federates!

Advance tag to (1 sec, 0) and invoke reaction.

Federated

c : Count

p : Print

(0, 1 sec)
Centralized Coordination: Next Event Request (NET)

No response yet because RTI cannot assure that all messages have been received.

Federated

\( c : \text{Count} \)
(0, 1 sec)

out
in

\( p : \text{Print} \)

NET(5 secs) (timeout time)
Upon completing execution at timeout time, each federate resigns.
Feedback with Centralized Coordination

Consider what happens a logical time 1 sec.

Blocks until either TAG(1 sec) is received or a message is received.

Message received - Executes

Executes

Executes

Message received - Executes

LTC: Logical Tag Complete
NET: Next Event Tag
TAG: Tag Advance Grant
PTAG: Provisional TAG

Blocks until either TAG(1 sec) is received or a message is received.
target C {
    coordination: decentralized
};

After clock synchronization, establish a direct socket connection and send messages.

RTI not involved after startup.

In this example, the default safe-to-process offset of zero mostly works because each federate can safely immediately process an event if it knows about that event.
EALMAC:c eal$ bin/Decentralized_p

...  
Federate 1: ERROR: STP violation occurred in a trigger to reaction 1, and there is no handler.  
**** Invoking reaction at the wrong tag!  
Federate 1: Received: 0 at (0, 1)  
Federate 1: Received: 1 at (1000000000, 0)  
Federate 1: Received: 2 at (2000000000, 0)  
Federate 1: Received: 3 at (3000000000, 0)  
Federate 1: Received: 4 at (4000000000, 0)  
Federate 1: ERROR: Received message too late. Already at stop tag. Current tag is (5000000000, 0) and intended tag is (5000000000, 0). Discarding message.

What happened?
Federate 1: Starting timestamp is: 1652852093838036000.
Federate 1: Timer ticked at (0, 0).
Federate 1: ERROR: STP violation occurred in a trigger to reaction 1, and there is no handler.
**** Invoking reaction at the wrong tag!
Federate 1: Received: 0 at (0, 1)
Federate 1: Timer ticked at (1000000000, 0).
Federate 1: ERROR: STP violation occurred in a trigger to reaction 1, and there is no handler.
**** Invoking reaction at the wrong tag!
Federate 1: Received: 1 at (1000000000, 1)
Federate 1: Timer ticked at (2000000000, 0).
Federate 1: ERROR: STP violation occurred in a trigger to reaction 1, and there is no handler.
**** Invoking reaction at the wrong tag!
Federate 1: Received: 2 at (2000000000, 1)
Federate 1: Timer ticked at (3000000000, 0).
...
With After

If after delay is greater than network delay, then no STP violations occur.

Federate 1: Starting timestamp is: 1652852276394596000.
Federate 1: Timer ticked at (0, 0).
Federate 1: Received: 0 at (10000000, 0)
Federate 1: Timer ticked at (1000000000, 0).
Federate 1: Received: 1 at (1010000000, 0)
Federate 1: Timer ticked at (2000000000, 0).
Federate 1: Received: 2 at (2010000000, 0)
Federate 1: Timer ticked at (3000000000, 0).
Federate 1: Received: 3 at (3010000000, 0)
Federate 1: Timer ticked at (4000000000, 0).
Federate 1: Received: 4 at (4010000000, 0)
Federate 1: Timer ticked at (5000000000, 0).
Federate 1 has resigned.
If STP offset is greater than network delay, then no STP violations occur.

Note: STP_offset here is STA in CAL paper (Safe To Advance).

Federate 1: Starting timestamp is: 1652852565042039000.
Federate 1: Received: 0 at (0, 0).
Federate 1: Timer ticked at (0, 0).
Federate 1: Received: 1 at (1000000000, 0).
Federate 1: Timer ticked at (1000000000, 0).
Federate 1: Received: 2 at (2000000000, 0).
Federate 1: Timer ticked at (2000000000, 0).
Federate 1: Received: 3 at (3000000000, 0).
Federate 1: Timer ticked at (3000000000, 0).
Federate 1: Received: 4 at (4000000000, 0).
Federate 1: Timer ticked at (4000000000, 0).
Federate 1: Received: 5 at (5000000000, 0).
Federate 1: Timer ticked at (5000000000, 0).
Federate 1 has resigned.
What is the difference?

**Inconsistency of 10 msec.**

**Unavailability of 10 msec.**
Unavailability of 10 msec.
CAL:

Consistency and/or Availability *must* be sacrificed as network Latency increases in any distributed system.
STP Violation Handlers

Inconsistency of 10 msec.

Unavailability of 10 msec.
Assumptions:

- **Deadlines**
  - WCET
  - Schedulability
- **Federated Execution (centralized)**
  - Reliable, in-order network (TCP/IP)
- **Federated Execution (decentralized)**
  - Deadlines
  - Network latency
  - Clock synchronization error
Contrast With...

- Publish and subscribe (e.g. ROS, MQTT)
- Actors (e.g. Akka, Ray)
- Shared memory
- Service-oriented architectures (RPC)
At What Cost Determinism?

- **Synchronized clocks**
  - These are becoming ubiquitous

- **Bounded network latency**
  - Violations are *faults*. They are detectable.

- **Bounded execution times**
  - Only needed in particular places.
  - Solvable with PRET machines (another talk).
Clock Synchronization

- NTP is widely available but not precise enough.
- IEEE 1588 PTP is widely supported in networking hardware but not yet by the OSs.
- Lingua Franca can work without clock synchronization by reassigning timestamps to network messages.
  - In this case, determinism is preserved within each multicore platform, but not across platforms.
Despite using TCP/IP on Ethernet, this network achieves highly reliable bounded latency.

TSN (time-sensitive networks) is starting to become pervasive...

This Bosch Rexroth printing press is a cyber-physical factory using Ethernet and TCP/IP with high-precision clock synchronization (IEEE 1588) on an isolated LAN.
Predicting the Future

Clock synchronization is going to change the world (again)

1500s days

1800s seconds

2000s nanoseconds

Gregorian Calendar (BBC history)

Lackawanna Railroad Station, 1907, Hoboken. Photograph by Alicia Dudek

2005: first IEEE 1588 plugfest

Lee, Berkeley
Global Positioning System

Provides ~100ns accuracy to devices with outdoor access.

Lee, Berkeley
It is routine for physical network interfaces (PHY) to provide hardware support for PTPs.

With this first generation PHY, clocks on a LAN agree on the current time of day to within ns, far more precise than GPS older techniques like NTP.
An Extreme Example: The Large Hadron Collider

The WhiteRabbit project at CERN is synchronizing the clocks of computers 10 km apart to within 10s of psec using a combination of GPS, IEEE 1588 PTP and synchronous ethernet.
Conclusions

• Lingua Franca programs are **testable** (timestamped inputs -> timestamped outputs)
• LF programs are **deterministic** under clearly stated assumptions.
• Violations of assumptions are **detectable** at run time.
• Actors, Pub/Sub, SoA, and shared memory have none of these properties.