182.703: Problems in Distributed Computing
(Part 1)
WS 2019

Ulrich Schmid
Institute of Computer Engineering, TU Vienna
Embedded Computing Systems Group E191-2
s@ecs.tuwien.ac.at

http://ti.tuwien.ac.at/ecs/teaching/courses/prdc
Target: Fault-tolerant Distributed RT Systems

Spatially distributed reactive computations

Real-time requirements

Partial failures

Worst-case response time $RT \leq T_{max}$
Interdisciplinary Research

REAL-TIME SYSTEMS

DIGITAL INTEGRATED CIRCUITS

FAULT-TOLERANT DISTRIBUTED ALGORITHMS
Motivation:
Distributed Fault-Tolerant Clock Generation in Systems-on-Chip
Clocking in Systems-on-Chip (I)

Classic synchronous paradigm

- **Concept:** Common notion of time for entire chip
- **Method:** Single crystal oscillator
  Global, phase-accurate clock tree

Disadvantages

- Cumbersome clock tree design (physical limits!)
- High power consumption
- Clock is **single point of failure!**
Clocking in Systems-on-Chip (II)

Alternative: DARTS clocks

- **Concept:** Multiple synchronized tick generators
- **Method:** Distributed FT tick generation algorithm
  Implemented in (asynchronous) HW

http://ti.tuwien.ac.at/ecs/research/projects/darts

Advantages
- Reasonable synchrony
- Uncritical clock distribution
- Clock is no single point of failure!
The DARTS Distributed Algorithm

On init
→ send $\text{tick}(0)$ to all; $C := 0$;

If got $\text{tick}(l)$ from $f + 1$ nodes and $l > C$
→ send $\text{tick}(C+1), \ldots, \text{tick}(l)$ to all;
    $C := l$;

If got $\text{tick}(C)$ from $2f + 1$ nodes
→ send $\text{tick}(C+1)$ to all;
    $C := C+1$;

For $n \geq 3f + 1$ and up to $f$ node failures, with (small) e-t-e delays $\in [d, d+\varepsilon]$:

• Suppose node $p$ sends $\text{tick}(C+1)$ at time $t$

• Then, node $q$ also sends $\text{tick}(C+1)$ by time $t+d+2\varepsilon$

⇒ Clock ticks occur approximately at the same time

2f + 1

f + 1

$p$ at $t$

any $q'$ at $t+\varepsilon$

any $q$ at $t+d+2\varepsilon$

$\leq \varepsilon$

$\leq d_{\text{max}} = d + \varepsilon$
$n \geq 3f+1$: Why do Failures hurt so much?

**Toy example:**

- A ("Byzantine" faulty)
- A: 08:00
- B: 10:00
- C: 08:00

$\Rightarrow$ 08:00

- A: 08:00
- B: 10:00
- C: 08:00

$\Rightarrow$ 08:00

- A: 10:00
- B: 10:00
- C: 08:00

$\Rightarrow$ 10:00

- With this algorithm, B and C never get closer together
- Will prove: Majority $n = 2f + 1$ not enough for $f$ Byz. failures!
Pipe Compare Signal Generators (PCSGs): There exists a dedicated
detection circuit for each pair of pipes which generates the status signals
$GEQ^{o/e}_{p,q}(t)$ and $GR^{o/e}_{p,q}(t)$. In particular, $GEQ^{o}_{p,q}(t')$ becomes active (i.e.,
$GEQ^{o}_{p,q}(t')$ becomes active (i.e.,
$GEQ^{o}_{p,q}(t')$ becomes active (i.e.,
$GEQ^{o}_{p,q}(t')$ becomes active (i.e.,
$GEQ^{o}_{p,q}(t')$ becomes active (i.e.,

**Definition 4.1.** (Direct Causality). Let $I(t')$ and $O(t)$ be two events of some
specific signal input and output, respectively, of a correct component $C$. Then
$I(t')$ and $O(t)$ are directly causally related, denoted by $I(t') \rightarrow O(t)$, if

(i) $r_{p,q}(t') \in N_{odd}$

(ii) $r_{p,q}(t') \not\equiv 0$

(iii) $r_{p,q}(t') \equiv 0$

Proof. First of all, establish for any correct component $C$

(bound)

$b^{\text{max}}(t')$

Assume that $b^{\text{max}}(t')$


**Theorem 4.13.** (Precision). The precision $\pi \geq |b_q(t) - b_p(t)|$ of our algo-

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$

$r_{p,q}(t) \equiv 0$
DARTS Implementation

Node p

Pipeline 1

Remote Pipe

Diff-Gate

Local Pipe

Remote Pipe

Diff-Gate

Local Pipe

 Pipeline 1

 Diff-Gate

 Pipeline 2

 Diff-Gate

 Pipeline 3

 Diff-Gate

 Pipeline 3f+1

 Pipe Compare Signal Gen.

 Threshold Logic

 Compare Signal Gen.

 Remote Pipe

 Diff-Gate

 Local Pipe

 Pipeline 3f+1

U. Schmid

182.703 PRDC
DARTS Extension: Self-Stabilization

SS Pulse Synchronization
Self-stabilizing, but moderate skew, low frequency

Tick Synchronization (DARTS)
Nominally low skew, high freq., but not self-stabilizing

force node reset
Introduction to Distributed Algorithms
Content (Part 1)

- Basics:
  - Distributed Computing Model
  - Synchrony and Fault-Tolerance
  - Correctness Proofs

- Some Appetizers:
  - Consistent Broadcasting
  - Consensus

- Food for Thoughts
Classic Modeling and Analysis

- Processors/processes modeled as interacting state machines

- **Zero-time** atomic computing steps, usually time-triggered
  - Message Passing (MP): [receive] + compute + [send]
  - Shared Memory (SHM): [accessSHM] + compute

- System timing parameters:
  - Operation durations modeled via **inter-step times** $\epsilon[\mu^-,\mu^+]$ (often $\mu^- = 0$)
  - Message delays modeled as **end-to-end delays** $\epsilon[\tau,\tau^+]$ (often $\tau = 0$)
Synchrony Models: 2 Extremes ...

Lock-step synchronous systems

- Computing step times:
  \[ \mu^- = \mu^+ = R \]
- Message delays
  \[ 0 \leq \tau \leq \tau^+ \leq R \]
- Perfectly synchronized rounds

Asynchronous systems

- Computing step times:
  - \( \mu^- = 0 \)
  - \( \mu^+ \) finite (but unbounded)
- Message delays
  - \( \tau = 0 \)
  - \( \tau^+ \) finite (but unbounded)
Failure Models

• „Deterministic“ failure models
  – At most $f$ of $n$ processors in the system may fail
  – Correct processes do not a priori know who has failed and when and how

• Failure semantics ranging from
  – Crash failures: Processors stop operating, possibly within a step
  – Byzantine failures [LSP82]: Processors can do what they want

• Real processors etc. fail probabilistically $\rightarrow$ Coverage analysis

• Restrict our attention to message passing systems here:
  – Typically fully connected, with dedicated links between every pair of processors
  – Receiver cannot be spoofed w.r.t. sender of a message
  – [Communication between correct processes typically considered reliable]
Message Passing vs. Shared Memory (I)

- MP can always be simulated in a SHM system.
- The opposite is not generally true:
  - Linearizable AsyncSHM can be simulated in AsyncMP only when a majority of processes do not crash \((n > 2f)\).
- MP is more elementary than SHM.
- SHM is more powerful than MP.

**Impossibility proof for** \(n \leq 2f\):
\[ p \in S_0, |S_0| = n/2, q \in S_1, |S_1| = n/2 \]

**Example:**
- \(\alpha_0\): \(R = 0, S_1\) dead
- \(\alpha_1\): \(R = 0, S_0\) dead
- \(\alpha_2\): \(R = 0\)
- **Merge** \(\alpha_0 \& \alpha_1\): Indistinguishable for \(S_0, S_1\)!
- \(\neg\)linearizable!
Message Passing vs. Shared Memory (II)

- **Wait-free** \((f = n-1)\) **event ordering** in AsyncSHM:
  - \(p\) knows (already by \(t_p\)) whether \(q\) has done some step!
  - \(p\) and \(q\) can **agree** on order of having done some step if no “in-between” crash occurs!

- **Impossible in AsyncMP!**

- **Uses „write-before-read“:**
  - \(p\) sets \(O[p] := 1\) if \(q\) has set \(R[q] := 1\)
  - Both \(O[p] := 0\) and \(O[q] := 0\) impossible
    - Event order \(p\) before \(q\) if \(O[p] = 0 \land O[q] = 1\) or \(O[p] = 1 \land O[q] = 1\)
    - Event order \(q\) before \(p\) if \(O[q] = 0 \land O[p] = 1\)
    - Event order **undecided (forever)** if either \(p\) or \(q\) crashes in between its two Writes

\[
\begin{align*}
R[p] &= 0, \quad O[p] = \perp \\
R[q] &= 0, \quad O[q] = \perp \\
\end{align*}
\]
Correctness Proofs

• Global state transitions
  – Configuration $C =$ vector of processor local states [+ in-transit messages for MP]
  – State transition = result of a single processor taking a step

• Algorithm vs. Adversary
  – Adversary determines which and when events $\varphi$ (like processor $p_i$ takes a step) happen ($\rightarrow$ Async. systems: Adv. subject to admissibility (fairness) conditions)
  – Algorithm determines what actually happens in the corresponding step

• Executions and traces
  – Execution $E =$ sequence of configurations alternating with events $C_0,\varphi_1,C_1,\varphi_2,C_2,\varphi_3,C_3, \ldots$
  – Trace $T =$ (sub-)sequence of „interesting“ events (or states)

• Correctness proofs: Set of generated traces satisfies
  – Safety properties („something bad never happens“)
  – Liveness properties („something good eventually happens“)
Some Appetizers
Consistent Broadcasting
Consistent Broadcasting [ST87]

- Want to build **authenticated reliable broadcasting**:  
  - Any process $p_s$ may have some message $m_s$ to broadcast: $\text{bcast}(p_s,m_s)$  
  - Every correct process shall eventually call $\text{accept}(p_s,m_s)$, and shall be sure that the received $m_s$ originates in $p_s$  
  - Do not use real authentication (cryptography)!

- Very useful primitive:  
  - Clock synchronization  
  - Consensus  
  - etc.
Properties Consistent Broadcasting

Time-free specification:

- **Correctness**: If a correct processor $p_s$ executes $\texttt{bcast}(p_s,m_s)$, then every correct processor eventually calls $\texttt{accept}(p_s,m_s)$

- **Unforgeability**: If a correct processor $p_s$ never executes $\texttt{bcast}(p_s,m_s)$, then no correct processor ever calls $\texttt{accept}(p_s,m_s)$

- **Relay**: If some correct processor calls $\texttt{accept}(p_s,m_s)$, then every other correct processor eventually also calls $\texttt{accept}(p_s,m_s)$
Implementation

\textbf{bcast}(p_s,m_s) at p_s

send \((init,p_s,m_s)\) to all processors

\textbf{accept}(p_s,m_s) at every \(p_i\)

\begin{align*}
\text{if got} & (init,p_s,m_s) \text{ from } p_s \\
& \rightarrow \text{send} (echo,p_s,m_s) \text{ to all [once]} \\
\text{if got} & (echo,p_s,m_s) \text{ from } f + 1 \\
& \rightarrow \text{send} (echo,p_s,m_s) \text{ to all [once]} \\
\text{if got} & (echo,p_s,m_s) \text{ from } 2f + 1 \\
& \rightarrow \text{call accept}(p_s,m_s)
\end{align*}

**System model:**

- At most \(f\) Byzantine faulty processors
- \(n \geq 3f + 1\)
- E-t-e delays \(\in [d,d+\varepsilon]\):

- Message sent by correct proc at \(t\) got by correct receiver proc within \([t+d,t+d+\varepsilon]\)
- Every proc gets at most \(f\) faulty echo messages from different proc
- At most \(f\) echo messages available at \(p_i\) by \(t\) could be missing at \(p_j\) by \(t + \varepsilon\)
Correctness Proof (Time-dependent Version)

- **Correctness:** If a correct proc $p_s$ executes $\text{bcast}(p_s,m_s)$ by $t$, then every correct processor eventually calls $\text{accept}(p_s,m_s)$ by $t+2(d+\varepsilon)$

- **Unforgeability:** If a correct proc $p_s$ does not execute $\text{bcast}(p_s,m_s)$ by $t$, then no correct processor calls $\text{accept}(p_s,m_s)$ by $t+2d$

- **Relay:** If a correct processor calls $\text{accept}(p_s,m_s)$ at $t$, then every other correct processor also calls $\text{accept}(p_s,m_s)$ by $t+d+2\varepsilon$

---

**Relay:**

- $p_i$ at $t$
- $\forall p_j' \text{ at } t+\varepsilon$
- Any $p_j$ at $t+d+2\varepsilon$

$2f+1$

$f+1$

$\leq \varepsilon$

$\leq d+\varepsilon$
A Note on Formal Verification

Problem: Correctness proofs for FT distributed algorithms (at most $f$ out of $n$ processes Byzantine faulty)

- State-of-the-art: Case-by-case “paper-and-pen” proofs, typically lacking rigorousness

  - **Exciting novel approach:** Apply finite-state parameterized model checking, based on advanced abstractions

Some nice results:

- Rigorous foundation: Interval and counter abstractions for threshold bounds: [JKSVW13b]
- Fast model-checking of Byzantine fault-tolerant atomic broadcast algorithms: [JKSVW13]
Consensus
A Classic Problem: Distributed Agreement (Consensus)
Consensus Properties

• Every process $p_i$
  – has initial value $x_i$ chosen from some finite set $V$
  – shall irrevocably decide on output value $y_i$

• **Termination**: Every correct processor eventually decides

• **Agreement**: Every two correct processors $p_i, p_j$ decide on the same value $y_i = y_j$

• **Validity**: If all correct processors have the same input value $x$, then $x$ is the only possible decision value
Asynchronous Consensus Impossibility

Fischer, Lynch and Paterson [FLP85]:

“There is no deterministic algorithm for solving consensus in an asynchronous distributed system in the presence of a single crash failure.”

Key problem:
Distinguish slow from dead!
Distributed Agreement (Consensus) - FLP

Yes

Yes

No

Yes

No

Yes

None meet

Yes

All meet

No
Synchronous Consensus

Lamport, Shostak and Pease [LSP82]:

“There is a deterministic algorithm for solving consensus in a synchronous distributed system of \(n \geq 3f+1\) processors in the presence of at most \(f\) Byzantine failures.”

But:
It is impossible to solve consensus if \(n = 3f\)!
Impossibility of Consensus for $f = 1$, $n = 3$

- Suppose correct algorithm $\mathcal{A} = (A,B,C)$ for $(p_0,p_1,p_2)$ existed

- Assume $p_0$ faulty
- By Validity:
  - $x_1 = x_2 = 0 \rightarrow y_1 = y_2 = 0$
  - $x_1 = x_2 = 1 \rightarrow y_1 = y_2 = 1$
- By Agreement:
  - $x_1 \neq x_2 \rightarrow y_1 = y_2$
„Easy Impossibility Proofs“ [FLM86] (I)

Arrange 6 correct processors in a ring:

Resulting execution will not solve consensus, but …
„Easy Impossibility Proofs“ [FLM86] (II)

Local view of $p_1, p_2$:

By Validity: Decision must be $y_1 = y_2 = 0 \ldots$
„Easy Impossibility Proofs“ [FLM86] (III)

Local view of $p_3, p_4$:

By Validity: Decision must be $y_3 = y_4 = 1$ …
„Easy Impossibility Proofs“ [FLM86] (IV)

Local view of \( p_2, p_3 \):

By Agreement: Decision should be \( y_2 = y_3 \) \( \rightarrow \) Contradicción
Food for Thoughts
Exercise: Consensus via Lossy Links

• Consider a synchronous system of n=2 processors that never fail, which are connected via a pair of directed links that may lose messages:

• Using a bivalence-proof, show that deterministic consensus is impossible even when, in every round, at most one (unknown) link may be lossy.

• Prove that consensus can be solved under any of the following two assumptions:
  i. The link $p \rightarrow q$ is not lossy in some round.
  ii. Exactly one of the links is lossy in every round.
The End

(Part 1)
References