Target: Fault-tolerant Distributed RT Systems

Spatially distributed reactive computations

Real-time requirements

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

Partial failures

U. Schmid

RiSE Winter School 2013
Interdisciplinary Research

REAL-TIME SYSTEMS

- deadlines
- Pressure
- preemption
- response times
- rate-monotonic
- jobs
- resources

FAULT-TOLERANT DISTRIBUTED ALGORITHMS

- leader election
- consensus
- reliable broadcast
- clock synchronization
- uniprocessor failures
- scheduling
- ParSync models
- metastability
- VHDL
- dependable architectures
- systems-on-chip
- clocking
- logic synthesis
- simulation
- EDP

DIGITAL INTEGRATED CIRCUITS

- dependable architectures
- systems-on-chip
- clocking
- asynchronous circuits
- logic synthesis
- simulation
- EDP

U. Schmid

RiSE Winter School 2013
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 tick(0) to all; C := 0;

If got tick(l) from f + 1 nodes and l > C
→ send tick(C+1),…, tick(l) to all;
    C := l;

If got tick(C) from 2f + 1 nodes
→ send 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 tick(C+1) at time \( t \)
- Then, node \( q \) also sends tick(C+1) by time \( t+d+2\varepsilon \)

⇒ Clock ticks occur approximately at the same time
n ≥ 3f + 1: Why do Failures hurt so much?

Toy example:

- 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 \( GEQ_{p,q}^{o/e}(t) \) and \( GR_{p,q}^{o/e}(t) \). In particular, \( GEQ_{p,q}^{o}(t') \) becomes active (i.e.,

\[
GEQ_{p,q}^{o}(t')
\]

previously

(i) \( r^s_{p,q}(t) \)
(ii) \( |r^p_{p,q}(t')| \)

Similar

(i) \( r^s_{p,q}(t) \in \mathbb{N}_{odd} \)
(ii) \( |r^p_{p,q}(t')| \)

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) they are
(ii) there is

\[
\text{Theorem 4.13. (Precision). The precision } \pi \geq |b_q(t) - b_p(t)| \text{ of our algorithm is bounded by } \pi \leq \left[ \frac{T_{syn}}{T_{first}} \right] + 1.
\]

Proof. First of all, establishment of a bound is established for a given \( k \), i.e., \( t^p_k \leq b^{max}(t') \)

\[
b^{max}(t') = \left[ \frac{\Delta}{T_{first}} \right] + \min \left\{ \pi + 1, \left[ \frac{\Delta}{D} - \frac{\Delta}{T_{first}} \right] \right\}.
\]

Theorem 4.14. (Accuracy). Given \( \Delta = t_2 - t_1 \), the accuracy \( |b_p(t_2) - b_p(t_1)| \) of any correct process \( p \) is bounded by \( \max \left\{ 0, \frac{\Delta - T_{syn} - T_{c}}{T_{first}} \right\} \leq |b_p(t_2) - b_p(t_1)| \leq \left[ \frac{\Delta}{T_{first}} \right] + \min \left\{ \pi + 1, \left[ \frac{\Delta}{D} - \frac{\Delta}{T_{first}} \right] \right\} \).

Proof. The upper bound for accuracy will be shown first: It is known that

\[
\forall t : b_p(t) \geq b^{max}(t) - \pi + (1 - I_{async}(t)) \quad \text{and} \quad \forall t : b_p(t) \leq b^{max}(t) + \pi - (1 - I_{async}(t)).
\]

Thus \( b_p(t_2) - b_p(t_1) \leq b^{max}(t_2) - b^{max}(t_1) + \pi - (1 - I_{async}(t_1)) \). By applying Lemma 4.11, \( b_p(t_2) - b_p(t_1) \leq \left[ \frac{\Delta}{T_{first}} \right] + \left[ 2I_{async}(t_1) - 1 + \pi \leq \left[ \frac{\Delta}{T_{first}} \right] + \pi + 1 \leq \left[ \frac{\Delta}{T_{first}} \right] + \pi + 1 \right] \). Moreover, from Lemma 4.7 it follows that \( b_p(t_2) - b_p(t_1) \leq \left[ \frac{\Delta}{D} \right] \). Hence, \( b_p(t_2) - b_p(t_1) \leq \left[ \frac{\Delta}{D} \right] \) or \( \left[ \frac{\Delta}{D} \right] \) is bounded.

To prove the lower bound, first define \( b_1 = b_p(t_1), b_2 = b_p(t_2) \) and \( t^p_1 \leq t_2, \ t^p_2 \leq t_2 \) as the points in time when \( p \) sends tick \( b_1 \) and \( b_2 \). Clearly \( t^p_{b_2+1} > t_2 \).
DARTS Implementation [FS11]
DARTS Extension: Self-Stabilization

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

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

force node reset

Ongoing research [DFLS11]
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


Nancy Lynch: *Distributed Algorithms*. Morgan Kaufmann, 1996
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

![Diagram of processes p and q with timing parameters μ⁻, μ⁺ and τ⁻, τ⁺]

- System timing parameters:
  - Operation durations modeled via **inter-step times** $\varepsilon[\mu^-,\mu^+]$ (often $\mu^- = 0$)
  - Message delays modeled as **end-to-end delays** $\varepsilon[\tau,\tau^+]$ (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
  – [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:
  - Linerarizable AsyncSHM can be simulated in AsyncMP only when a majority of processes \((n > 2f)\) do not crash
- MP is more elementary than SHM
- SHM is more powerful than MP

**Impossibility proof for** \(n \leq 2f\):

\[
p \in S_0, |S_0| = n/2, q \in S_1, |S_1| = n/2
\]

\[
\alpha_0: \begin{cases} 
    R=0, S_1 \text{ dead} \\
    t_0
\end{cases}
\]

\[
\alpha_1: \begin{cases} 
    R=0, S_0 \text{ dead} \\
    t_0 \\
    t_1
\end{cases}
\]

\[
\text{Write}_p R := 1
\]

\[
\text{Read}_q R = 0
\]

Merge \(\alpha_0 \& \alpha_1\): Indistinguishable for \(S_0, S_1\)!

\[
\alpha_2: \begin{cases} 
    R=0 \\
    t_1
\end{cases}
\]

\[
\text{Write}_p R := 1
\]

\[
\text{Read}_q R = 0
\]

\[\neg \text{linearizable!}\]

U. Schmid

RiSE Winter School 2013

18
Message Passing vs. Shared Memory (II)

- Wait-free ($f = n-1$) event ordering in AsyncSHM:
  - $p$ knows whether $q$ has already 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 $O[p] := 0 \land O[q] := 1$
    - 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

\[
\begin{align*}
  & p \quad \text{Write}_p R[p] := 1 \\
  & q \quad \text{Write}_q R[q] := 1 \\
  & \quad \text{Read}_p R[q] \\
  & x = \text{Read}_q R[p] \\
  & \quad \text{Write}_p O[p] := x \\
  & \quad \text{Write}_q O[q] := x
\end{align*}
\]
Correctness Proofs

- Global state transitions
  - Configuration $C = \text{vector of processor local states} [+ \text{in-transit messages for MP}]$
  - 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 ($\rightarrow \text{Async. systems: Adv. subject to admissibility (fairness) conditions}$)
  - Algorithm determines what actually happens in the corresponding step

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

- 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 (\( \rightarrow \) DARTS!)
  - Consensus
  - etc.
Properties Consistent Broadcasting

Time-free specification:

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

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

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

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

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

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

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

System model:

\begin{itemize}
\item At most \(f\) Byzantine faulty processors
\item \(n \geq 3f + 1\)
\item E-t-e delays \(\in [d,d+\varepsilon]\):
\item Message sent by correct proc at \(t\) got by correct receiver proc within \([t+d,t+d+\varepsilon]\)
\item Every proc gets at most \(f\) faulty echo/init messages from different procs
\item At most \(f\) echo messages available at \(p_i\) by \(t\) could be missing at \(p_j\) by \(t + \varepsilon\)
\end{itemize}
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 $t$
- any $p_j'$ at $t+\varepsilon$
- $\leq \varepsilon$
- $\leq d+\varepsilon$
- any $p_j$ at $t+d+2\varepsilon$
A Note on Formal Verification

• Typical distributed algorithms proofs are definitely „handwaving“, compared to verification standards

• Making proofs amenable to theorem-proving is tedious [SWR02]

• Model checking is challenging, even for simple problems like CB:
  – Parameterization: How to handle not just an unspecified number of processes, but rather both unspecified \((n, f)\) ?
  – Failures: How to exhaustively incorporate allowed faulty behaviors?
  – Synchrony assumptions: How to deal with asynchronous/synchronous/timed/partially synchronous systems?

• We are working on this in the context of RiSE:
Consensus
A Classic Problem: Distributed Agreement (Consensus)

Yes

Yes

No

Yes

No

Yes

None meet

All meet

No

No
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

➢ **Consensus is instrumental for decision making in non-centralized distributed systems**
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

Yes

Yes

Yes

Yes

Yes

Yes

Yes

None meet

All meet

No

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 $A = (A,B,C)$ 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 $y_1 = y_2 = 0$ …
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 $y_2 = y_3 \rightarrow$ Contradiction
Food for Thoughts
Communciation Failures

- **Link failure model:**
  1. Distinguish send and receive link failures
  2. Distinguish omission and arbitrary link failures
  3. Indep. for every send/rec to/from all

- **Known results:**
  - $n > f^r_l + f^s_l$ necessary & sufficient for solving consensus with pure link omission failures
  - $n > f^r_l + f^{ra}_l + f^s_l + f^{sa}_l$ necessary & sufficient for solving consensus with link omission and arbitrary failures
Exercises

1. Find the smallest values for $S,R,S',R',S'',R''$ in the CB implement below for arbitrary link failures ($f_i^r = f_i^{ra}$ and $f_i^s = f_i^{sa}$):

   \[
   \begin{align*}
   &\text{if} \text{ got } (\text{init}, p_s, m_s) \text{ from } p_s \\
   &\quad \rightarrow \text{send } (\text{echo}, p_s, m_s) \text{ to all [once]} \\
   &\text{if} \text{ got } (\text{echo}, p_s, m_s) \text{ from } Sf_i^{sa} + Rf_i^{ra} + f + 1 \\
   &\quad \rightarrow \text{send } (\text{echo}, p_s, m_s) \text{ to all [once]} \\
   &\text{if} \text{ got } (\text{echo}, p_s, m_s) \text{ from } S'f_i^{sa} + Rf_i^{ra} + 2f + 1 \\
   &\quad \rightarrow \text{call } \text{accept}(p_s, m_s)
   \end{align*}
   \]

   Required number of procs:
   - $n \geq S''f_i^{sa} + R''f_i^{ra} + 3f + 1$

   Link failure lower bound:
   - $n \geq f_i^r + f_i^{ra} + f_i^s + f_i^{sa}$

2. Find an „easy impossibility proof“ that shows that $n=4$ processors are not enough for solving consensus with $f_i^r = f_i^{ra} = f_i^s = f_i^{sa} = 1$ (and $f = 0$)
The End

(Part 1)
References