Distributed Algorithms
(Part 1)
RiSE Winter School 2012

Ulrich Schmid
Institute of Computer Engineering, TU Vienna
Embedded Computing Systems Group E182/2
s@ecs.tuwien.ac.at
Target: Fault-tolerant Distributed RT Systems

Spatially distributed reactive computations

Real-time requirements

Partial failures

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

REAL-TIME SYSTEMS RESEARCH
- deadlines
- Pressure Sensor
- response times
- rate-monotonic
- jobs
- resources

DISTRIBUTED ALGORITHMS RESEARCH
- leader election
- reliable broadcast
- clock synchronization
- ParSync model
- message passing
- hybrid systems
- LTL

VERIFICATION RESEARCH
- computing steps
- reactive systems
- model checking
- theorem proving
- SAT
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

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 \( t+d+2\varepsilon \)

\[ \Rightarrow \text{Clock ticks occur approximately at the same time} \]

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

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

If got \( \text{tick}(C) \) from \( 2f+1 \) nodes
- \( \rightarrow \) send \( \text{tick}(C+1) \) to all;
  \( C := C+1; \)
$n \geq 3f+1$: 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 $GEQ_{p,q}^{\text{o/e}}(t)$ and $GR_{p,q}^{\text{o/e}}(t)$. In particular, $GEQ_{p,q}^o(t')$ becomes active (i.e., $GEQ_{p,q}^o(t') = 1$).

**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) $r_{p,q}^{\text{self}}(t) \in \mathbb{N}_{\text{odd}}$ and

(ii) $r_{p,q}^{\text{self}}(t) \in \mathbb{N}_{\text{even}}$.

**Theorem 4.13.** (Precision). The precision $\pi \geq |b_q(t) - b_p(t)|$ of our algorithm is bounded by $\pi \leq \left\lceil \frac{T_{\text{sum}}}{\overline{T}_{\text{first}}} \right\rceil + 1$.

**Proof.** First, we establish for any process $p$ that $\pi \geq b^\text{max}(t)$.

Assume that prior

**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 $\max\left\{0, \frac{\Delta - T_{\text{sum}} - T^+}{D^+}\right\} \leq |b_p(t_2) - b_p(t_1)| \leq \left\lceil \frac{\Delta}{\overline{T}_{\text{first}}} \right\rceil + \min\left\{\pi + 1, \left\lceil \frac{\Delta}{D^+} - \frac{\Delta}{\overline{T}_{\text{first}}} \right\rceil \right\}$. The upper bound for accuracy will be shown first: It is known that $\forall t : b_p(t) \geq b^\text{max}(t) - \pi + (1 - I_{\text{async}}(t))$ and $\forall t : b_p(t) \leq b^\text{max}(t)$ from Lemma 4.13 and Lemma 4.11. Thus $b_p(t_2) - b_p(t_1) \leq b^\text{max}(t_2) - b^\text{max}(t_1) + \pi - (1 - I_{\text{async}}(t_1))$. By applying Lemma 4.11, $b_p(t_2) - b_p(t_1) \leq \left\lceil \frac{b - b_1}{\overline{T}_{\text{first}}} \right\rceil + 2I_{\text{async}}(t_1) - 1 + \pi \leq \left\lceil \frac{b - b_1}{\overline{T}_{\text{first}}} \right\rceil + \pi + 1 \leq \left\lceil \frac{b - b_1}{\overline{T}_{\text{first}}} \right\rceil + \pi + 1$. Moreover, from Lemma 4.7 it follows that $b_p(t_2) - b_p(t_1) \leq \left\lceil \frac{b - b_1}{D^+} \right\rceil$. Hence, $b_p(t_2) - b_p(t_1) \leq \min\left\{\left\lceil \frac{\Delta}{\overline{T}_{\text{first}}} \right\rceil + \pi + 1, \left\lceil \frac{\Delta}{D^+} \right\rceil \right\} \leq \left\lceil \frac{\Delta}{\overline{T}_{\text{first}}} \right\rceil + \min\left\{\pi + 1, \left\lceil \frac{\Delta}{D^+} - \frac{\Delta}{\overline{T}_{\text{first}}} \right\rceil \right\} \right\} \leq \left\lceil \frac{\Delta}{\overline{T}_{\text{first}}} \right\rceil + \min\left\{\pi + 1, \left\lceil \frac{\Delta}{D^+} - \frac{\Delta}{\overline{T}_{\text{first}}} \right\rceil \right\} \right\}$. Since $[x + y] \leq [x] + [y]$.

To prove the lower bound, first define $b_1 = b_p(t_1)$, $b_2 = b_p(t_2)$ and $t_{b_1}^o \leq t_2$, $t_{b_2}^o \leq t_2$ as the points in time when $p$ sends tick $b_1$ and $b_2$. Clearly $t_{b_2+1}^o > t_2$.  

DARTS Implementation

U. Schmid

RiSE Winter School 2012

11
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** $\epsilon[\delta^-,\delta^+]$ (often $\delta^- = 0$)
Synchrony Models: 2 Extremes …

**Lock-step synchronous systems**

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

**Asynchronous systems**

- Computing step times:
  \[ \mu^- = 0 \]
  \[ \mu^+ \text{ finite (but unbounded)} \]
- Message delays
  \[ \delta^- = 0 \]
  \[ \delta^+ \text{ finite (but unbounded)} \]
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]
A Note on Message Passing vs. Shared Memory

- MP can always be simulated in a SHM system
- The opposite is not generally true:
  - AsyncSHM can be simulated in AsyncMP if a majority of processes \((n > 2f)\) is correct
  - Not the case for \(n \leq 2f\) \(\Rightarrow\) AsyncSHM more powerful than AsyncMP
- MP is more elementary than SHM!

- E.g.: Wait-free \((f = n-1)\) event ordering possible in AsyncSHM but not in AsyncMP

\[\text{p knows by the time of its Read whether q has already done its Write}\]
Correctness Proofs

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

• 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 $E = \text{sequence of configurations alternating with events}$
    $C_0, \varphi_1, C_1, \varphi_2, C_2, \varphi_3, C_3, \ldots$
  – 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
  – 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

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

send \texttt{(init},p_s,m_s) \text{ to all processors}

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

\texttt{if} got \texttt{(init},p_s,m_s) \text{ from } p_s
\rightarrow \text{ send } \texttt{(echo},p_s,m_s) \text{ to all } \text{[once]}

\texttt{if} got \texttt{(echo},p_s,m_s) \text{ from } f + 1
\rightarrow \text{ send } \texttt{(echo},p_s,m_s) \text{ to all } \text{[once]}

\texttt{if} got \texttt{(echo},p_s,m_s) \text{ from } 2f + 1
\rightarrow \text{ call } \texttt{accept}(p_s,m_s)

\textbf{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/init messages from different pros
- 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 $t+d+2\varepsilon$

\[ \leq \varepsilon \quad \leq d + \varepsilon \]

U. Schmid

RiSE Winter School 2012
Verification Challenges

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

• Try do make it rigorous is challenging, even for simple problems like CB:
  – Parameterization \((n, f)\)
  – Asynchronous systems
  – Failures

• We are working on this in the context of RiSE …
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

Yes

No

Yes

Yes

No

Yes

Yes

No

Yes

Yes

No

Yes

Yes

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:
  - $x_1 \neq x_2 \rightarrow y_1 = y_2$
Arrange 6 **correct** processors in a ring:

Resulting execution will not solve consensus, but …
By Validity: Decision must be $y_1 = y_2 = 0$ ...
„Easy Impossibility Proofs“ [FLM86] (III)

Local view of $p_3, p_4$:

By Validity: Decision must be $y_3 = y_4 = 1$ ...
Local view of $p_2, p_3$:

By Agreement: Decision should be $y_2 = y_3 \rightarrow$ Contradiction
Food for Thoughts
Communcation 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:
  - $n > f_i^r + f_i^s$ necessary & sufficient for solving consensus with pure link omission failures
  - $n > f_i^r + f_i^{ra} + f_i^s + f_i^{sa}$ 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 implem. below for arbitrary link failures ($f^r_l = f^{ra}_l$ and $f^s_l = f^{sa}_l$):

   if got $(init, p_s, m_s)$ from $p_s$
          → send $(echo, p_s, m_s)$ to all [once]
   if got $(echo, p_s, m_s)$ from $S f^sa_l + R f^{ra}_l + f + 1$
          → send $(echo, p_s, m_s)$ to all [once]
   if got $(echo, p_s, m_s)$ from $S' f^sa_l + R' f^{ra}_l + 2 f + 1$
          → call accept$(p_s, m_s)$

   Required number of procs:
   • $n \geq S'' f^sa_l + R'' f^{ra}_l + 3 f + 1$

   Recall lower bound:
   • $n \geq f^r_l + f^{ra}_l + f^s_l + f^{sa}_l + 3 f + 1$

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

References


References

References