Fault-Tolerant Distributed Algorithms

RiSE Winter School 2013 (Part 3)

Ulrich Schmid
Embedded Computing Systems Group E182/2
s@ecs.tuwien.ac.at
Content (Part 3)

- Distributed [FT] Real-Time Systems
  - Introduction to Real-Time Scheduling
  - Reconciling Distributed and Real-Time Systems
  - Non-Synchronous Distributed Real-Time Systems
  - Real-Time Research Challenges …

- VLSI Systems-on-Chip vs. Distributed Systems
  - Modeling framework
  - Fault-modeling challenges
Distributed [FT] Real-Time Systems
Recall: [Fault-tolerant] Distributed RT Systems

Spatially distributed reactive computations

Real-time requirements

Partial failures

Worst-case response time $RT \leq T_{\text{max}}$
Recall Classic DC 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 process interactions](image)

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

- DC research established a wealth of results:
  - Correctness proofs of distributed algorithms
  - Impossibility & lower bound results
Real-Time Properties?

**Classic modeling:**
- \([\mu^-,\mu^+], [\tau, \tau^+]\) are *a priori given* system parameters (alg-indep.)
- Analysis considers occurrence times of steps *independently* of each other:
  - No queueing & scheduling in the picture
  - Too optimistic time complexity

**Reality:**
- \([\mu^-,\mu^+], [\tau, \tau^+]\) depend on algorithms + scheduling policies
- Non-preemptible operations \(\rightarrow\) steps not independent:
  - Time complexity analysis involves *real-time analysis*
  - Moser & Schmid [MS06,MS08]
Fixed Step Times in SHM Systems?

Process $p$

Process $q$

Process $r$

Access $R$

$[\mu, \mu^+]$ depends on contention!
Fixed End-to-End Delays in MP Systems?

Op time $[\mu^-,\mu^+]$

Transm. delay $[\delta^-,\delta^+]$

E-t-e delay $[\tau^-,\tau^+]$ depends on receiver load!
Real-Time Analysis (RTA) Framework (MP)

RTA challenge:
• Determine $F(.)$, by identifying best & worst case scenarios
• Determine $[\tau^-, \tau^+]$, by solving equation $[\tau^-, \tau^+] = F(A[\tau, \tau^+], S)$

RTA yields: $[\tau^-, \tau^+] = F(A([\tau, \tau^+]), S)$

"Educated guess": $[\tau^-, \tau^+] = [10^{-\mu}, 100 \delta^+]$

Alg $A$ determines: Message pattern, computing load,
Computing system $S = S([\mu^-, \mu^+], [\delta^-, \delta^+])$

System $S$ determines:
Computing & communication speeds, scheduling
Real-Time Analysis Challenges

- **Queuing phenomena:**
  - Multiple messages from different processors
  - Interrupt handling
  - Multitasking at processors
  - Multiple messages per link

- **End-to-end delays** $\tau^+$ hence depend on
  - Message & computational complexity of algorithms!
  - Scheduling disciplines!
  - External event/interrupt load!
  - Task interaction (precedence relations, critical sections)
  - Failures
Consequences

- \( \tau^+ \) can be huge in real systems, since the processing of all messages must be taken into account
  - Maximum determines synchronous round duration \( \rightarrow \) overkill for most messages
  - Partial escape: Use messages classes & scheduling algorithms.

- **Example**: Failure Detector-based Algorithms using Fast FDs [HL02]
  - Use Head-of-the-Line Scheduling for FD-level processes and messages
  \( \Rightarrow \tau^+ \) relevant for FD algorithm reduced by orders of magnitude, and much easier to determine
Introduction to Real-Time Scheduling
Real-Time Computing (I)

• Central problem: **Real-Time Scheduling** of
  – tasks on processors
  – messages on communication channels
  – occupancy of mutual exclusive resources

• Example algorithms:
  – Rate Monotonic Scheduling (RM)
  – Earliest Deadline First (EDF)

• Goals:
  – Guaranteed response times
  – Good utilization of resources
Real-Time Computing (II)

- Real-time computing is well-established field [SAAC04]:
  - Wealth of results for uniprocessor scheduling
  - Many results for multiprocessor scheduling
  - Some results for loosely-coupled distributed systems


Note: The intro slides for RM and EDF have been adopted from Insup Lee’s CIS700 course at UPenn: [http://www.cis.upenn.edu/~lee/05cis700/slides_ppt/lec07-real-time-scheduling.ppt](http://www.cis.upenn.edu/~lee/05cis700/slides_ppt/lec07-real-time-scheduling.ppt)
Task Characteristics ($n$ Tasks $\tau_1, \ldots, \tau_n$)

- **Task $\tau_i$ execution properties:**  
  - Worst-case execution time $C_i$ (WCET)  
  - Criticality level  
  - Preemptive / non-preemptive execution  
  - Task can / cannot be suspended during execution

- **Task $\tau_i$ completion constraints:**  
  - Relative deadline $D_i$:  
    - Hard: Completion by the deadline mandatory  
    - Firm: Completion by the deadline or no execution  
    - Soft: Best-effort completion  
  - Jitter: Completion within some time interval

- **Task $\tau_i$ release constraints:**  
  - Periodic tasks: Released periodically with period $T_i$  
  - Sporadic tasks: Minimum inter-release time $T_i$  
  - Aperiodic tasks: No constraints

- **Task $\tau_i$ dependencies:**  
  - Precedence relations among tasks  
  - Resource sharing during task execution (shared data, communication links, …)
Real-Time Scheduling

• Schedule execution of a set of tasks
  • on the processor and resources available in the system
  • such that all timing constraints are met
  • \(\rightarrow\) Need a feasible schedule

• Static (off-line) scheduling has a priori knowledge of
  – characteristic parameters of all tasks
  – all future task release times (e.g. periodic tasks)
  \(\triangleright\) Typically allows guarantees and optimal algorithms

• Dynamic (on-line) scheduling
  – Only knows currently active task set
  – does not know future task releases
  \(\triangleright\) Guarantees and optimality more difficult to establish [if at all existent]
Schedulability

• Property indicating whether a real-time system (a set of real-time tasks) can meet all deadlines

• Example: $n=3$ periodic tasks $\tau_i (T_i, C_i)$ with $D_i = C_i$, $1 \leq i \leq 3$

Can we schedule this on a single processor?
Real-Time Scheduling Algorithms

- Determines the order of real-time task executions
- Major classes:
  - Static-priority scheduling (RM)
  - Dynamic-priority scheduling (EDF)
RM (Rate Monotonic) Scheduling

• SPS with priority assigned according to period:
  – A task with a shorter period has a higher priority
  – Always execute pending job with the shortest period

• Optimal static-priority scheduling algorithm
RM (Rate Monotonic) Scheduling

- Always executes pending job with the shortest period
RM (Rate Monotonic) Scheduling

- Always executes pending job with the shortest period

\[
\begin{align*}
\tau_1 \ (4,1) \\
\tau_2 \ (5,2) \\
\tau_3 \ (7,2)
\end{align*}
\]
RM – Utilization Bound [LL73]

- A set of periodic/sporadic tasks is schedulable under RM if

\[ \sum C_i/T_i \leq n \left(2^{1/n} - 1 \right) \]
EDF (Earliest Deadline First) Scheduling

- Dynamic priority algorithm:
  - A task with a shorter deadline has a higher priority
  - Always executes pending task with the earliest deadline

- Optimal algorithm (except under overload)
EDF (Earliest Deadline First) Scheduling

- Always executes pending task with the earliest deadline
EDF (Earliest Deadline First) Scheduling

- Always executes pending task with the earliest deadline
EDF (Earliest Deadline First) Scheduling

- Always executes pending task with the earliest deadline

\[ \tau_1 (4,1) \]
\[ \tau_2 (5,2) \]
\[ \tau_3 (7,2) \]
EDF (Earliest Deadline First) Scheduling

- Optimal scheduling algorithm
  - if there is a feasible schedule for a set of real-time tasks, EDF can schedule it.

\[ \tau_1 (4,1) \]
\[ \tau_2 (5,2) \]
\[ \tau_3 (7,2) \]
EDF – Utilization Bound [LL73]

• A set of periodic/sporadic tasks is schedulable under EDF if and only if

\[
\sum \frac{C_i}{T_i} \leq 1
\]

• Note: Tasks with
  – critical sections
  – precedence constraints
can also be handled by means of EDF (using modified deadlines).
EDF under Overload

- **Domino effect** during overload conditions
  - Example: $\tau_1(4,3)$, $\tau_2(5,3)$, $\tau_3(6,3)$, $\tau_4(7,3)$ all pending at $t = 0$

```
Deadline Miss!
```

![Better schedules diagram]

```
Better schedules:
```

U. Schmid  
RiSE Winter School 2013
RM vs. EDF [But05]

• Rate Monotonic
  – Simple implementation, even in systems without explicit support for timing constraints (periods, deadlines)
  – Predictability only for the highest priority tasks
  – Low processor utilization

• EDF
  – Full processor utilization
  – Optimal in the absence of overload
  – Bad performance under overload
Response Time Analysis

- Response time $RT_i$
  - Duration from release time to finishing time of task $\tau_i$
Response Time Analysis

- Response time $RT_i$
  - Duration from release time to finishing time of task $\tau_i$
Response Time Analysis for RM

- Response Time $RT_i$ for RM [Audsley et al., 1993]

$$RT_i = C_i + \sum_{\tau_k \in HP(\tau_i)} \left[ \frac{RT_i}{T_k} \right] \cdot C_k$$

$HP(\tau_i)$: a set of higher-priority tasks than $\tau_i$
Distributed Real-Time Systems?

• Response time analysis can be extended to incorporate release jitter:
  – Additional (but non-cumulative) delay between release of a task and time it can be scheduled first
  – Can be used to model variability of message delivery delays at receiving processor, as caused by
    • Sending processor CPU scheduling
    • Message scheduling in the network

• **Holistic Schedulability Analysis**
  – Static priority scheduling: [TC94]
  – EDF: [Spu96]
Reconciling Distributed and Real-Time Systems
Real-Time Distributed Computing Model

- **RT model core features** [Moser & Schmid, OPODIS’06]

- Investigate relation classic vs. RT model
  - Carry over classic failure models?
  - Carry over classic correctness proofs?
  - Carry over classic time complexity results?
  - Carry over classic impossibility & lower bound results?

- Conduct real-time analysis for e-t-e delays $\tau^+$
State-Transition Problems

Can be defined for both models in the same way:

State-transition problem = a set of state-transition traces
Example: Problem Definition

Deterministic Drift-Free Clock Synchronization

\( is\_finalstate(g) :\Leftrightarrow \forall g' > g : \forall p : s_p(g) = s_p(g') \)

**Termination**: All processors eventually terminate.

\( \exists g : is\_finalstate(g) \)

**Agreement**: After all processors have terminated, all processors have adjusted clocks within \( \gamma \) of each other.

\( \forall g : is\_finalstate(g) \Rightarrow (\forall p, q : |AC_p(g) - AC_q(g)| \leq \gamma) \)
Example: Drift-Free Clock Sync

Classic Model:

Result:
- Optimal worst-case precision
- Optimal running time $O(1)$

Real-time Model:

First try:
Direct transformation
(conduct real-time analysis)

Result: Poor performance
- Sub-optimal worst-case precision
- $O(n)$ time

Result:
- Optimal worst-case precision
- Achievable only in time $O(n)$
- $O(1)$ time algorithm with sub-optimal precision also exists
Non-Synchronous Distributed RT-Systems
State of the Art

• Today’s belief: FT-RT system must be synchronous:
  – known bounds on computing step times
  – known transmission delay bounds
  – known bounds on clock drifts

• Sync assumptions OK 99% of the time, but real systems
  – depend on physics and other sources of uncertainty
  – may experience unanticipated op conditions and overload

• Moreover, worst-case scenarios are rare events, hence
  – difficult to determine
  – yield systems with poor resource utilization
Example: Replicated Servers

Employ Byzantine Agreement/Atomic Broadcasting:

Clients send request messages to servers using BA/ABC

Agreement: Servers must agree on received messages to guarantee replica consistency

Validity: Servers must eventually get all messages from correct clients [eventual message delivery]

Timeliness: Message delivery must occur within time $\Delta$
Synchronous BA/ABC

Replica consistency could be violated if some $\delta > R - \pi$!

- Synchronous BA/ABC algorithm
  - Agreement guaranteed if $\delta \leq R - \pi$
  - Validity guaranteed if $\delta \leq R - \pi$
  - Timeliness guaranteed if $\delta \leq R - \pi$

- Simulate lock-step rounds of duration $R \geq \tau^+ + \pi$, where $\tau^+$ is max. e.t.e.-delay of high-level msgs

- Build $\pi$-synchronized clocks using low-level msgs with e.t.e.-delay $\tau^+_{\text{low}}$
Alternatives to Synchronous Systems?

- Do all essential properties of a FT-RT system need Sync?
  
  **NO!**

- Are there ways to guarantee logical safety & liveness properties independently of the timing properties of the underlying system?
  
  **YES: Asynchronous algorithms**

- Are there suitable time-free computational models and algorithms?
  
  **YES: E.g. ABC/Θ-Model**
Synchronous BA/ABC atop ABC/Θ-Model

Replica consistency not violated if $\delta > \tau^+$, as long $\Theta$ OK!

- Synchronous BA/ABC algorithm
  - Agreement guaranteed if $\Theta$ holds
  - Validity guaranteed if $\Theta$ holds
  - Timeliness guaranteed if $\delta \leq \tau^+$

- Simulate lock-step rounds of duration $R \geq 3\Theta$ local clock ticks

- Build synchronized clocks using $\Theta$-clock synchronization algorithm

Synchronous BA/ABC algorithm

Lock-step rounds

Synchronized clocks

Hardware
Nevertheless:

- $\tau^+$ can be huge in real systems since all messages [including application-level] must be taken into account
  - Maximum determines synchronous round duration $\rightarrow$ too conservative for most messages
  - Partial escape: Use messages classes & scheduling algorithms

  - Use Head-of-the-Line Scheduling for FD-level processes and messages
  - Only blocking factors due to non-preemptible resources can lead to priority inversion phenomena at low-level

$\Rightarrow \tau^+$ relevant for low-level FD algorithm reduced by orders of magnitude, and much easier to determine
Asynchronous BA/ABC with FDs

Replica consistency NEVER violated since indep. of $\delta$!

- Async FD-based BA algorithm
  - **Unconditional** Agreement
  - Validity requires FD semantics OK
  - Timeliness requires bounded $\delta$

- Implement FD, e.g. using low-level messages only

[HL02]
Real-Time Analysis of FastFDs:

Hermant & Le Lann [HLL02]: \( \tau^+ = \gamma(n) \) with

\[
\gamma(x) = w_{outQ} + w_{outq} + \delta_m + \left( \log_m(l) + x + \xi^l_x \right) \sigma + \left[ x(1 - \frac{\sigma}{w_{inqu}}) \right] w_{inqu}
\]

\[
\xi^l_x = m \left[ \log_m(m \left\lfloor \frac{x}{2} \right\rfloor) \right] - 1 + m \left\lfloor \frac{x}{2} \right\rfloor \left[ \log_m \left( \frac{1}{m \left\lfloor \frac{x}{2} \right\rfloor} \right) \right] - (x - m \left\lfloor \frac{x}{2} \right\rfloor), x \in \{2, ..., l\}
\]

(Note that \( w_{outQ}, w_{outq} \) and \( w_{inqu} \) are the problematic parts here)

- Would you trust a real system to always obey this, during the whole mission time?
- Would you really want your safety and liveness properties to depend on this?

Not needed!
Summary: Properties of Async BA

• Timeliness properties inevitably need Sync
  – $\delta \leq R - \pi$ for Sync BA; $R$ determined by worst-case $\tau^+$
  – Bounded $\delta$ for Async BA

  $\Rightarrow$ **Async BA better average performance**!

• Agreement and Validity do not need Sync
  – Validity achievable without bounded $\delta$
  – Agreement achievable even without FD semantics OK

  $\Rightarrow$ Async BA always guarantees replica consistency

  $\Rightarrow$ **Async BA higher assumption coverage**!
Real-Time Research Challenges …
Summary: The DC – RT Modeling Challenge

Classic modeling

Zero-time actions

$Proc\ p$

Trigger events (e.g. msg arrival)

Zero-time state transitions

Perfect synchrony:

$R \geq \tau^+$

Non-preemptible operations:

Queueing, Scheduling

Non-zero-time state transitions

Non-preemptible operations:

$[\mu, \mu^+]$

Asynchrony:

$\tau^+ > \tau^+$

$\tau^+ < \tau^+$

Reality

Too optimistic time complexity bounds

Synchrony costly to build

Assumption coverage?

Performance penalty

U. Schmid

RiSE Winter School 2013
Research Goals

• Convince DC computing people of the importance of RTA, by contradicting results of traditional analysis
• Establish methods for RTA of synchronous networked distributed systems
• Establish methods for RTA of ParSync networked distributed systems
RT Analysis of Lock-step Round Algorithms

• Every round \( k \) consists of
  – a computing + send step [or maybe a sequence of such steps] at process
    \( p \) that compute and broadcast message \( m^k_p \), based on the round \( k-1 \)
    messages received by the beginning of the round
  – Receive steps for every arriving message, making the message content
    accessible for the next computing step

• All round \( k \) computing steps at correct processes are released at
  exactly the same time [triggered periodically every \( R \) seconds,
  e.g. via perfectly synchronized local clocks]

• Very simple model, widely known in DC community and easily
  simulated in the classic synchronous model by using
  approximately synchronized clocks
Timing of Lock-step Round Execution

- **Process p**
  - Compute $m^k_r$
  - $q$ gets $m^k_r$

- **Process q**
  - Compute $m^k_r$
  - $q$ gets $m^k_r$

- **Process r**
  - Compute $m^k_r$

$E$-t-e. delay $\delta^r_q$

$x(k-1)$  $\mathbf{R}$ (round $k$)  $x(k)$  $\mathbf{R}$ (round $k+1$)  $x(k+1)$

U. Schmid  RiSE Winter School 2013
Target 1: RT Analysis Lock-step Consensus

- Analyze lock-step Byzantine generals algorithm (EIG) from [LSP82], under some reasonable scheduling strategy.

- "Recursive" $f+1$ round algorithm: In round $k$ (starting at $f+1$),
  - General process $i$ broadcasts local value $x_i$ in a round, which is received as $x_i^j$ at process $j$.
  - For value $x_i^j$ received at process $j$, spawn a new $k$-1 round Byzantine generals instance to disseminate agreed $y_i^j$.
  - Use majority of $y_i^j$ as agreed $y_i$.

- Ideal candidate for our purpose:
  - Perfectly synchronous rounds at all processes make periodic scheduling theory applicable.
  - Tolerates Byzantine failures $\Rightarrow$ allows to analyze the effects of different failure assumptions on end-to-end delays.
  - The number of concurrent instances increases with every round $\Rightarrow$ will lead to different $\tau^+$ for different rounds because increasing load!
Target 2: RT Analysis of DLS Consensus

- Analyze Byzantine consensus algorithm from [DLS88] [with $\Phi=1$, known $\Delta$ and unknown GST] under some reasonable scheduling strategy.

- Goal is to compute worst-case termination time RT after GST, which
  - will involve solving fixed-point equations for $\Delta$
  - $\Delta$ and hence RT will depend on GST due to increasing message size for compensating message loss $\rightarrow$ the simple DLS model is contradictory.

- Advantages
  - Tolerates Byzantine failures $\rightarrow$ allows to analyze the effects of different failure assumptions (= adversaries) on end-to-end delays!
  - $\Phi=1$ implies lock-step synchronous execution and hence enables lock-step rounds
RT Analysis of non-Lock-step Algorithms

• Consider algorithms executing in asynchronous rounds
• Every process $r$’s round $k$ consists of
  – a computing and a send step [or a sequence such steps] at process $r$ that compute and broadcast message $m^k_r$, based on $n-f$ round $k-1$ messages received at $r$ by that time
  – Receive steps for every arriving message, making the message content accessible for the next computing step and [optionally] triggering the next round computations
• Round $k$ computing steps at different processors are released at (very) different times!
• Widely known in DC community
• Requires additional assumptions [e.g. DLS or failure detectors or Theta-Model] for solving consensus.
Target 3: RT Analysis of ABC Clock Sync

• Analyze ABC Byzantine clock synchronization algorithm from [RS08] under some reasonable scheduling strategy.
  – Integer clock value = current round number
  – Single type of messages only, of bounded size via wrap-around of round numbers

• Goal is to compute worst-case precision and accuracy, which
  – involves solving fixed-point equations for minimal and maximal end-to-end delay $\Delta^-$ and $\Delta^+$
  – Needs simultaneous worst-case & best-case scheduling analysis, without periodic task releases!

• Advantages
  – Relatively simple message patterns
  – Tolerates Byzantine failures $\rightarrow$ allows to analyze the effects of different failure assumptions (= adversaries) on end-to-end delays!
  – Is building block for perfect FD
Timing of Round-based Async. Execution

- Process $p$
  - Computation step
  - Reception

- Process $q$
  - Computation step
  - Reception
  - $q$ gets $m_r^k$

- Process $r$
  - Computation step
  - Reception
  - Computation transaction
  - Reception
  - $n - f$
  - E-t-e. delay $\delta_q^r$

- $x_r(k - 1)$
- $r$'s round $k$
- $x_r(k)$
- $r$'s round $k + 1$
- $x_r(k + 1)$

U. Schmid

RiSE Winter School 2013
Tools for RT Analysis?

- Holistic analysis [Spuri 1996]
- Trajectory approach [Martin, Minet & George 2005]
- Network calculus [Boudec & Tiran, 2003]
- Min/Max + algebra
- Something else?
- Something new?
An Example

• Asynchronous round synchronization algorithm
  [Robinson & Schmid, SSS’08]
  – Processes switch to next round $k+1$ when $n-1$ messages from previous round $k$ arrived
  – Also extends to failures [wait for $n-f$ messages]
  – Time complexity is determined by time series of round switching times $x_p(k)$

On init
  → send round(1) to all;

If got round($k$) from $n-1$ procs
  → send round($k+1$) to all;
Max-Plus Algebra (I)

Let

• $x_p(k) =$ start of round $k$ on process $p$
• $\delta_{ij} =$ message delay of message from $i$ to $j$

$$x_3(k+1) = \max(x_1(k) + \delta_{13}, x_2(k) + \delta_{23})$$
\( \mathcal{R}_{\text{max}} = (\mathbb{R}_{\text{max}}, \oplus, \otimes, \epsilon, e) \)

with

\[
\begin{align*}
\epsilon & := -\infty \\
\epsilon & := 0 \\
\mathbb{R}_{\text{max}} & := \mathbb{R} \cup \{\epsilon\} \\
a \oplus b & := \max(a, b); \quad a, b \in \mathbb{R}_{\text{max}} \\
a \otimes b & := a + b; \quad a, b \in \mathbb{R}_{\text{max}}
\end{align*}
\]

yields

\[
x_3(k + 1) = (x_1(k) \otimes \delta_{13}) \oplus (x_2(k) \otimes \delta_{23})
\]
Max-Plus Algebra (III)

• Transmission delay matrix

\[ \Delta = \begin{pmatrix}
\epsilon & \delta_{12} & \cdots & \delta_{1n} \\
\delta_{21} & \epsilon & \cdots & \delta_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
\delta_{n1} & \delta_{n2} & \cdots & \epsilon
\end{pmatrix} \]

Round vector

\[ \vec{x}(k) = \begin{pmatrix}
x_1(k) \\
x_2(k) \\
\vdots \\
x_n(k)
\end{pmatrix} \]

• Max-plus matrix multiplication yields

\[ \vec{x}(k) = (\Delta^T)^{\otimes k} \otimes \vec{x}(0) \]
Making Things Difficult

• Need to incorporate
  – non-zero time steps
  – failures (round switching upon reception of $n$-$f$ msgs)

• Need Min-Max-Plus Algebra …
ParSync RTA Challenge

Recent idea: Min-max-plus algebra, game-theoretic modeling

$t_3(k+1) = \min(\max(t_1(k)+\delta_{13}+2\mu, t_2(k)+\delta_{23}+\mu), \max(t_1(k)+\delta_{13}+\mu, t_2(k)+\delta_{23}+2\mu))$
VLSI Systems-on-Chip vs. Distributed Systems
VLSI Circuits
Well-Known Trends in Chip Technology

- Shrinking feature size
- Increasing complexity
- Increasing clock speed
- Interconnect delays dominate switching delays
- Signals cannot traverse the chip within a single clock cycle
- Signals cannot traverse the entire chip within a single clock cycle
- Increasing susceptibility to transient failures (particles, cross-talk, ...)
- High power-consumption
- Combine fault-tolerant research to address those issues
- Integrate distributed computing and VLSI
Classic Distributed Computing Approach

**Proof goals:**

1. **Prove that the model meets the specification**
   - On booting:
     - Send `tick(0)` to all; `C := 0`;
     - If got `tick(X)` from `f+1` procs and `X > C`:
       - Send `tick(C+1),... ,tick(X)` to all [once];
     - `C := X`;
   - If got `tick(C)` from `n-f` processes:
     - Send `tick(C+1)` to all [once];
     - `C := C+1`;

2. **Minimize ‚„proof gap“ between model and implementation**

**Executable machine code, real system**

**N CS Algs, max. f Byz.**
- max precision
  - min/max frequency

**Synced FT clocks**
- Distributed state machine,
  - Byzantine failures

**TTP implementation**

**Proof**

**specification**

**abstraction**

**model (alg+sys)**

**implementation**

**SW**
DC Approach Applied to VLSI Circuits?

But what about the large proof gap?

On boot:
- Send $\text{tick}(0)$ to all;
- $C := 0$;
- If got $\text{tick}(X)$ from $f+1$ processes and $X > C$:
  - Send $\text{tick}(C+1), \ldots, \text{tick}(X)$ to all \[once\];
  - $C := X$;
- If got $\text{tick}(C)$ from $n-f$ processes:
  - Send $\text{tick}(C+1)$ to all \[once\];
  - $C := C+1$;

But what about the large proof gap?

if $[r_{p,q}(t) \geq r^*_{p,q}(t)] \land [r^*_{p,q}(t) \in \text{odd}] \land [s_{p,q}(t) = 1]$
  \[send GEQ^o_{p,q}(t) - \]
else \[send GEQ^o_{p,q}(t) - \]
if $[r_{p,q}(t) > r^*_{p,q}(t)] \land [r^*_{p,q}(t) \in \text{odd}] \land [s_{p,q}(t) = 1]$
  \[send GR^o_{p,q}(t) - \]
else \[send GR^o_{p,q}(t) - \]

if $GEQ^o_{p,q}(t)$ for at least $2f+1$ remote processes $q_i$
  \[send TH^o_{GEQ} - \]
else \[send TH^o_{GEQ} - \]
if $GR^o_{p,q}(t)$ for at least $f+1$ remote processes $q_i$
  \[send TH^o_{GR} - \]
else \[send TH^o_{GR} - \]
if $GEQ^o_{p,q}(t)$ for at least $2f+1$ remote processes $q_i$
  \[send TH^o_{GEQ} - \]
else \[send TH^o_{GEQ} - \]
if $GR^o_{p,q}(t)$ for at least $f+1$ remote processes $q_i$
  \[send TH^o_{GR} - \]
else \[send TH^o_{GR} - \]

if $TH^o_{GR}(t) \land \neg[TH^o_{GR}(t) \lor TH^o_{GEQ}(t)]$
  \[send tick - \]
if $TH^o_{GR}(t) \lor TH^o_{GEQ}(t)$ \[\lor TH^o_{GR}(t) \lor TH^o_{GEQ}(t)]$
  \[send tick - \]
FATAL Modeling Framework [FS12]

**Modules:** Finite (typically hierarchical) composition of two basic elements:

- Zero-time **boolean functions** (AND, OR, …)
- FIFO **Channels** with continuous delay \( \delta(t) \in [\tau, \tau^+] \)

\[
o(t) = i(t - \delta(t))
\]

**Signal = continuum many events!**

- Purely digital (0,1) signals
- Continuous computations!
- Arbitrary environments
- Can model Zeno-behavior
Fault Modeling: Metastability
2. Metastability as Failure?

Bistable element (memory cell) with positive feedback

\[ u_{i,2} = u_{o,1} \]

stable (HI)

\[ u_{i,1} = u_{o,2} \]

stable (LO)

metastable
Revisit Muller C-Gate

Non-zero delays

Normal operation

Limited output slope

Oscillation

Creeping
What about Error Containment?

According to the proofs by Függer & Schmid [FS12], the wall holds for Byzantine failures – but this excludes metastability!
Fault Modeling: Radiation-induced Transient Failures
Particle Sources in the Universe

• Sources of cosmic radiation generate high energy (GeV-TeV) charged particles (electrons, protons, nuclei):
  – Rotating neutron stars (pulsars)
  – Supernovae
  – Double star systems
  – Galactic centers, black holes
  – Extragalactic sources (quasars)

• Generation/acceleration mechanisms:
  – Acceleration of charged particles in time-varying magnetic fields („cyclotron mechanisms“)
  – Shock wave acceleration, by particle reflection at fast shock waves (e.g. Supernovae explosions)
Radiation-induced Transient Failures

Soft error rates dominate in VLSI!

SET \rightarrow SEU
Effects of SETs on Asynchronous Circuits

Example: SEU-tolerant ring oscillator

For simplicity:
* Delay-free C-Gates
* Delay-free wires

Normal operation:

Faulty operation: Pulse injected
The End
(Part 3)
References