Unmasking Fault Tolerance: Masking vs. Non-masking Fault-tolerant Systems

Nils Müllner

nils.muellner@informatik.uni-oldenburg.de
Abteilung Systemsoftware und verteilte Systeme
Department für Informatik
Carl von Ossietzky Universität Oldenburg

February 22, 2011
Orientation
Outline

1. Motivation
2. Basics
3. Computation of LWAV
4. Lumping
5. Decomposition
6. Status and Outlook
Intel: Palisades

Resilient Processors: Self-Tuning Core
Circuit Research Lab, Hillsboro, Oregon

Enhancing energy efficiency through dynamic variation tolerance
- Detect and correct errors due to dynamic variations
- Eliminate guardbands to improve energy & performance
- Processor “self-tunes” to adapt to any environment

[Energy vs. Throughput]
- Conventional
- Resilient: 22% energy reduction at the same performance
- Resilient: 41% higher performance at the same energy

Research at Intel
Presented by Intel Labs Europe

[BTL+10]
Focus: Basic Research

- fault tolerance in distributed systems
  is important for a variety of systems like CPU, WSN, ...
Focus: Basic Research

- fault tolerance in distributed systems is important for a variety of systems like CPU, WSN, ...

- focus: not system specific fault tolerance methods, but fundamental principles.
Focus: Basic Research

- fault tolerance in distributed systems is important for a variety of systems like CPU, WSN, ...

- focus: not system specific fault tolerance methods, but fundamental principles.

⇒: relation between quality (degree of masking) and cost.
1 Motivation

2 Basics

3 Computation of LWAV

4 Lumping

5 Decomposition

6 Status and Outlook
Outline

1. fault tolerance demands redundancy
2. fault tolerance classification
3. the fault masker concept
4. unmasking fault tolerance
5. redundancy classification
6. self-stabilization
Fault Tolerance Demands Redundancy

- to **tolerate** faults, they must be detected and/or corrected
- detection and correction are functions that require resources
- typically either space (functional or information redundancy) or time (but commonly both)
- sometimes convertible (e.g., TMR)
Fault Tolerance Demands Redundancy

- to tolerate faults, they must be detected and/or corrected
- detection and correction are functions that require resources
- typically either space (functional or information redundancy) or time (but commonly both)
- sometimes convertible (e.g., TMR)

Example: **Cyclic Redundancy Checks (CRC)** requires space (extends the package, information redundancy), and more space (code for the computation of CRC, functional redundancy), and time (for the computation, and transmission, temporal redundancy)
Three Kinds of FT.: Focus on Masking and Non-masking

<table>
<thead>
<tr>
<th></th>
<th>safe</th>
<th>not safe</th>
</tr>
</thead>
<tbody>
<tr>
<td>live</td>
<td>masking</td>
<td>non-masking</td>
</tr>
<tr>
<td>not live</td>
<td>failsafe</td>
<td>intolerant</td>
</tr>
</tbody>
</table>

**Table:** Fault Tolerance Classes [KA97, Gär99]
Three Kinds of FT.: Focus on Masking and Non-masking

<table>
<thead>
<tr>
<th></th>
<th>safe</th>
<th>not safe</th>
</tr>
</thead>
<tbody>
<tr>
<td>live</td>
<td>masking</td>
<td>non-masking</td>
</tr>
<tr>
<td>not live</td>
<td>failsafe</td>
<td>intolerant</td>
</tr>
</tbody>
</table>

↑ correctors

Table: Fault Tolerance Classes [KA97, Gär99]
Three Kinds of FT.: Focus on Masking and Non-masking

<table>
<thead>
<tr>
<th></th>
<th>safe</th>
<th>not safe</th>
<th>detectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>live</td>
<td>masking</td>
<td>non-masking</td>
<td></td>
</tr>
<tr>
<td>not live</td>
<td>failsafe</td>
<td>intolerant</td>
<td></td>
</tr>
</tbody>
</table>

↑ correctors

Table: Fault Tolerance Classes [KA97, Gär99]

non-masking fault tolerance

• requires correction

• relatively cheap
Three Kinds of FT.: Focus on Masking and Non-masking

<table>
<thead>
<tr>
<th>live</th>
<th>not live</th>
</tr>
</thead>
<tbody>
<tr>
<td>safe</td>
<td></td>
</tr>
<tr>
<td></td>
<td>not safe</td>
</tr>
</tbody>
</table>

↑ correctors

Table: Fault Tolerance Classes [KA97, Gär99]

non-masking fault tolerance
- requires correction
- relatively cheap

masking fault tolerance
- requires detection and correction
- most desirable
Three Kinds of FT.: Focus on Masking and Non-masking

<table>
<thead>
<tr>
<th>live</th>
<th>masking</th>
<th>non-masking</th>
</tr>
</thead>
<tbody>
<tr>
<td>not live</td>
<td>failsafe</td>
<td>intolerant</td>
</tr>
</tbody>
</table>

Table: Fault Tolerance Classes [KA97, Gär99]

- non-masking fault tolerance
  - requires correction
  - relatively cheap

- masking fault tolerance
  - requires detection and correction
  - most desirable

- non-/masking fault tolerant with regards to a distinct fault class
Example: CRC

- intolerant: corrupted packet contained matching checksum
- non-masking fault tolerant: faults were detected, but could not be corrected / re-request violates temporal boundaries
- masking: correct transmission / or faults could be corrected on the spot
Example: CRC

- intolerant: corrupted packet contained matching checksum
- non-masking fault tolerant: faults were detected, but could not be corrected / re-request violates temporal boundaries
- masking: correct transmission / or faults could be corrected on the spot

non-/masking fault tolerant with regards to a distinct fault class
The Fault Masker

![Diagram showing the Fault Masker system]

system
user

request

fault
masker

system

\[ k \]

\( t \)

service unavailable
service available

[MDT09]
The Fault Masker

![Diagram of fault masker system]

request

system
user

fault
masker

system

\[ MDT09 \]
The Fault Masker

system user

fault masker

system

request

k

service unavailable

√ service available

\[\text{[MDT09]}\]
The Fault Masker

\[ \text{system user} \]

\[ \text{fault masker} \]

\[ \text{system} \]

\[ \text{request} \]

\[ k \quad k+1 \]

\[ \text{service unavailable} \quad \text{service available} \]

[MDT09]
The Fault Masker

[MDT09]
The Fault Masker

[MDT09]
The Fault Masker

[MDT09]
The Fault Masker

[MDT09]
The Fault Masker

[MDT09]
The Fault Masker

[MDT09]
The Fault Masker

\[ \text{[MDT09]} \]
The Fault Masker

The fault masker detects all faults

[MDT09]
Detection ⇒ Safety, Correction ⇒ Liveness

- failsafe
- intolerant

- masking
- nonmasking
Detection ⇒ Safety, Correction ⇒ Liveness

- **failsafe**
- **intolerant**
- **detectors**
- **correctors**
- **masking**
- **nonmasking**

Unmasking Fault Tolerance: Masking vs. Non-masking Fault-tolerant Systems
Detection ⇒ Safety, Correction ⇒ Liveness

- **failsafe**
- **fault masker**
- **detectors**
- **fault intolerant**
- **correctors**
- **nonmasking**
- **masking**
Detection $\Rightarrow$ Safety, Correction $\Rightarrow$ Liveness
Redundancy Establishes Detection and Correction

- information redundancy
  - error correcting or detecting codes
  - N-Modular Redundancy

- temporal Redundancy
  - self-stabilization
  - re-requests
  - N-Modular Redundancy
Focus: Correction Based on Temporal Redundancy (e.g., Self-Stabilization)

information redundancy: thoroughly discussed

- we can compute the quality of spatial redundancy (i.e., number and severity of faults covered, either in a masking or non-masking fashion)
- spatial redundancy commonly used to ensure data integrity
Focus: Correction Based on Temporal Redundancy (e.g., Self-Stabilization)

information redundancy: thoroughly discussed

- we can compute the quality of spatial redundancy (i.e., number and severity of faults covered, either in a masking or non-masking fashion)
- spatial redundancy commonly used to ensure data integrity

temporal redundancy (assuming detection as given):

- commonly used for system integrity
- how good can time heal/cure the system from faults?
- what is a proper metric?
- how can we calculate this metric?
Self-Stabilization

Definition (Self-Stabilization [Dol00, Dij74])

A system is self-stabilizing wrt. a safety predicate $P$ iff:

1. Starting from any state, it is guaranteed that the system will eventually reach a state that satisfies the safety predicate $P$ (convergence property), provided that no fault happens.

2. Given that the system satisfies the safety predicate, it is guaranteed to stay in a state that satisfies the safety predicate $P$ (closure property), provided that no fault happens.
<table>
<thead>
<tr>
<th>Motivation</th>
<th>Basics</th>
<th>Computation of $LWAV$</th>
<th>Lumpng</th>
<th>Decomposition</th>
<th>Status and Outlook</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td><strong>Computation of $LWAV$</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Suitable Metric 1/2

Definition (Limiting Window Availability (LWA))

Assume that at time $t = 0$, an initial system state distribution holds that corresponds to the steady state distribution of a system. Then, Limiting Window Availability of window size $w$ (of this system), denoted by $LWA_w$, $w \geq 0$, is the probability that the system has at least once reached a state satisfying $\mathcal{P}$ within the following $w$ time steps:

$$LWA_w = \text{prob}\{\exists k, 0 \leq k \leq w : c_k \models \mathcal{P}\}$$

$w$ is called window size.
A Suitable Metric 2/2

Definition (Limiting Window Availability Vector (LWAV))

The limiting window availability vector of size $i$ (of a system), denoted by $\text{LWAV}_i$, is an $i$-dimensional vector of probabilities. The element in the $i^{th}$ position is the limiting window availability of window size $i - 1$ of that system:

$$\text{LWAV}_i := \langle \text{LWA}_0, \text{LWA}_1, \ldots, \text{LWA}_{i-1} \rangle.$$
A Suitable Metric 2/2

Definition (Limiting Window Availability Vector (LWAV))

The limiting window availability vector of size $i$ (of a system), denoted by $\text{LWAV}_i$, is an $i$-dimensional vector of probabilities. The element in the $i^{th}$ position is the limiting window availability of window size $i - 1$ of that system:

$$\text{LWAV}_i := \langle \text{LWA}_0, \text{LWA}_1, \ldots, \text{LWA}_{i-1} \rangle.$$

Definition (Limiting Window Availability Vector Gradient (LWAVG))

The limiting window availability vector gradient of size $i$ (of a system), denoted by $\text{LWAVG}_i$, is an $i$-dimensional vector of probabilities. The element in the $i^{th}$ position is the limiting window availability of window size $i$ minus the limiting window availability of window size $i - 1$ of that system:

$$\text{LWAVG}_i := \langle \text{LWA}_1 - \text{LWA}_0, \text{LWA}_2 - \text{LWA}_1, \ldots, \text{LWA}_i - \text{LWA}_{i-1} \rangle.$$
Test Set-Up: Algorithm and Topology

\[
\text{const id} := 0,
\text{var reg},
\text{repeat}\{\text{reg} := 0\}
\]

\text{Figure: Broadcast Sub-Algorithm for the Root Process}

\[
\text{const neighbors} := \{\pi_i, \ldots\},
\text{const id} := \min\{\text{id}(\pi_i), \ldots\} + 1,
\text{var reg},
\text{var set} := \{\text{reg}_i, \pi(\text{reg}_i) \in \text{neighbors} | \forall i: \text{id}(\pi_i) = \text{id} - 1\}
\text{repeat}\{
\neg((\exists \text{reg}_i: \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 2)
\lor (\exists \text{reg}_i: \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 0))
\rightarrow \text{reg} := 1
\land \exists \text{reg}_i: \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 0
\rightarrow \text{reg} := 0
\land \exists \text{reg}_i: \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 2
\rightarrow \text{reg} := 2
\}
\]

\text{Figure: Broadcast Sub-Algorithm for Non-Root Processes}
Test Set-Up: Algorithm and Topology

\[
\text{const } \text{id} := 0, \\
\text{var } \text{reg}, \\
\text{repeat}\{ \\
\quad \text{reg} := 0 \}
\]

**Figure:** Broadcast Sub-Algorithm for the Root Process

\[
\text{const } \text{neighbors} := \{\pi_i, \ldots\}, \\
\text{const } \text{id} := \min\{\text{id}(\pi_i), \ldots\} + 1, \\
\text{var } \text{reg}, \\
\text{var } \text{set} := \{\text{reg}_i, \pi(\text{reg}_i) \in \text{neighbors} | \forall i: \text{id}(\pi_i) = \text{id} - 1\} \\
\text{repeat}\{ \\
\quad \neg((\exists \text{reg}_i: \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 2) \\
\quad \lor(\exists \text{reg}_i: \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 0)) \\
\quad \rightarrow \text{reg} := 1 \\
\quad \exists \text{reg}_i: \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 0 \\
\quad \rightarrow \text{reg} := 0 \\
\quad \exists \text{reg}_i: \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 2 \\
\quad \rightarrow \text{reg} := 2 \}
\]

**Figure:** Broadcast Sub-Algorithm for Non-Root Processes
Test Set-Up: Algorithm and Topology

\[
\text{const } \text{id} := 0, \\
\text{var } \text{reg}, \\
\text{repeat}\{ \\
\quad \text{reg} := 0 \}
\]

**Figure:** Broadcast Sub-Algorithm for the Root Process

\[
\text{const } \text{neighbors} := \{\pi_i, \ldots\}, \\
\text{const } \text{id} := \min\{\text{id}(\pi_i), \ldots\} + 1, \\
\text{var } \text{reg}, \\
\text{var } \text{set} := \{\text{reg}_i, \pi(\text{reg}_i) \in \text{neighbors}| \forall i : \text{id}(\pi_i) = \text{id} - 1\} \\
\text{repeat}\{ \\
\quad \neg ((\exists \text{reg}_i : \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 2) \\
\quad \lor (\exists \text{reg}_i : \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 0)) \\
\quad \rightarrow \text{reg} := 1 \\
\quad \square \exists \text{reg}_i : \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 0 \\
\quad \rightarrow \text{reg} := 0 \\
\quad \square \exists \text{reg}_i : \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 2 \\
\quad \rightarrow \text{reg} := 2\}
\]

**Figure:** Broadcast Sub-Algorithm for Non-Root Processes

Nils Müllner
Unmasking Fault Tolerance: Masking vs. Non-masking Fault-tolerant Systems
Test Set-Up: Algorithm and Topology

\[ \text{const id} := 0, \]
\[ \text{var reg,} \]
\[ \text{repeat} \{
    \text{reg} := 0
\} \]

**Figure:** Broadcast Sub-Algorithm for the Root Process

\[
\text{const neighbors} := \{\pi_i, \ldots\},
\text{const id} := \min\{\text{id}(\pi_i), \ldots\} + 1,
\text{var reg,}
\text{var set} := \{\text{reg}_i, \pi(\text{reg}_i) \in \text{neighbors} | \forall i : \text{id}(\pi_i) = \text{id} - 1 \}
\text{repeat} \{
\neg((\exists \text{reg}_i : \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 2) \\lor \neg(\exists \text{reg}_i : \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 0))
\rightarrow \text{reg} := 1
\}
\]
\[
\exists \text{reg}_i : \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 0
\rightarrow \text{reg} := 0
\]
\[
\exists \text{reg}_i : \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 2
\rightarrow \text{reg} := 2 \}
\]

**Figure:** Broadcast Sub-Algorithm for Non-Root Processes
Test Set-Up: Algorithm and Topology

```
const id := 0,
var reg,
repeat{
    reg := 0
}
```

**Figure:** Broadcast Sub-Algorithm for the Root Process

```
const neighbors := \{\pi_i, \ldots\},
const id := \min\{id(\pi_i), \ldots\} + 1,
var reg,
var set := \{reg_i, \pi(reg_i) \in neighbors | \forall i : id(\pi_i) = id - 1\}
repeat{
    \neg((\exists reg_i : \pi(reg_i) \in set \land reg_i = 2) \lor (\exists reg_i : \pi(reg_i) \in set \land reg_i = 0)) 
    \rightarrow reg := 1
    \exists reg_i : \pi(reg_i) \in set \land reg_i = 0 
    \rightarrow reg := 0
    \exists reg_i : \pi(reg_i) \in set \land reg_i = 2 
    \rightarrow reg := 2
```

**Figure:** Broadcast Sub-Algorithm for Non-Root Processes
Test Set-Up: Algorithm and Topology

```plaintext
const id := 0,
var reg,
repeat {
  reg := 0
}
```

Figure: Broadcast Sub-Algorithm for the Root Process

```plaintext
const neighbors := \{\pi_i, \ldots\},
const id := \min\{id(\pi_i), \ldots\} + 1,
var reg,
var set := \{reg_i, \pi(reg_i) \in neighbors | \forall i : id(\pi_i) = id - 1\}
repeat {
  \neg ((\exists reg_i : \pi(reg_i) \in set \land reg_i = 2) \lor (\exists reg_i : \pi(reg_i) \in set \land reg_i = 0))
  \rightarrow reg := 1
  \exists reg_i : \pi(reg_i) \in set \land reg_i = 0
  \rightarrow reg := 0
  \exists reg_i : \pi(reg_i) \in set \land reg_i = 2
  \rightarrow reg := 2
}
```

Figure: Broadcast Sub-Algorithm for Non-Root Processes
Test Set-Up: Algorithm and Topology

const id := 0,
var reg,
repeat{
    reg := 0}

Figure: Broadcast Sub-Algorithm for the Root Process

const neighbors := \{\pi_i, \ldots\},
const id := \min\{id(\pi_i), \ldots\} + 1,
var reg,
var set := \{reg_i, \pi(reg_i) \in neighbors|\forall i:id(\pi_i) = id-1\}
repeat{
    \neg((\exists reg_i:\pi(reg_i) \in set \land reg_i = 2) \lor (\exists reg_i:\pi(reg_i) \in set \land reg_i = 0)) \rightarrow reg := 1
    \square \exists reg_i:\pi(reg_i) \in set \land reg_i = 0 \rightarrow reg := 0
    \square \exists reg_i:\pi(reg_i) \in set \land reg_i = 2 \rightarrow reg := 2}

Figure: Broadcast Sub-Algorithm for Non-Root Processes
Test Set-Up: Algorithm and Topology

\[
\text{const } \text{id} := 0, \\
\text{var } \text{reg}, \\
\text{repeat}\{ \\
\quad \text{reg} := 0 \}
\]

Figure: Broadcast Sub-Algorithm for the Root Process

\[
\begin{array}{c}
\pi_1 \\
\pi_2 \\
\pi_3 \\
\pi_4
\end{array}
\]

\[c_t \models P: \text{reg}_1 = 0 \land \\
\quad \text{reg}_2 = 0 \land \\
\quad \text{reg}_3 = 0 \land \\
\quad \text{reg}_4 = 0
\]

\[
\begin{array}{c}
\text{const } \text{neighbors} := \{\pi_i, \ldots\}, \\
\text{const } \text{id} := \min\{\text{id}(\pi_i), \ldots\} + 1, \\
\text{var } \text{reg}, \\
\text{var } \text{set} := \{\text{reg}_i, \pi(\text{reg}_i) \in \text{neighbors}\mid \forall i: \text{id}(\pi_i) = \text{id} - 1\} \\
\text{repeat}\{ \\
\quad \neg((\exists \text{reg}_i: \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 2) \\
\quad \text{xor}(\exists \text{reg}_i: \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 0)) \\
\quad \rightarrow \text{reg} := 1 \\
\quad \square \exists \text{reg}_i: \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 0 \\
\quad \rightarrow \text{reg} := 0 \\
\quad \square \exists \text{reg}_i: \pi(\text{reg}_i) \in \text{set} \land \text{reg}_i = 2 \\
\quad \rightarrow \text{reg} := 2 \}
\]

Figure: Broadcast Sub-Algorithm for Non-Root Processes
### The Resulting Markov Chain

<table>
<thead>
<tr>
<th>from/to</th>
<th>(\langle 0, 0, 0, 0 \rangle)</th>
<th>(\langle 0, 0, 0, 2 \rangle)</th>
<th>(\langle 0, 0, 2, 0 \rangle)</th>
<th>(\langle 0, 2, 0, 0 \rangle)</th>
<th>(\langle 2, 0, 0, 0 \rangle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\langle 0, 0, 0, 0 \rangle)</td>
<td>(p(e_1 + e_2 + e_3 + e_4))</td>
<td>(qe_4)</td>
<td>(qe_3)</td>
<td>(qe_2)</td>
<td>(qe_1)</td>
</tr>
<tr>
<td>(\langle 0, 0, 0, 2 \rangle)</td>
<td>(pe_4)</td>
<td>(pe_3)</td>
<td>(pe_2)</td>
<td>(pe_1)</td>
<td>(pe_4)</td>
</tr>
<tr>
<td>(\langle 0, 2, 0, 0 \rangle)</td>
<td>(pe_3)</td>
<td>(pe_2)</td>
<td>(pe_1)</td>
<td>(pe_1)</td>
<td>(pe_1)</td>
</tr>
<tr>
<td>(\langle 2, 0, 0, 0 \rangle)</td>
<td>(pe_1)</td>
<td>(pe_1)</td>
<td>(pe_1)</td>
<td>(pe_1)</td>
<td>(pe_1)</td>
</tr>
</tbody>
</table>

**Table:** Transitions Grouped by Number of Operational Processes

- \(e_i\): probability, that \(\pi_i\) is elected for execution
- \(q\): probability, that a fault occurs
- \(p = 1 - q\)
- \(e_1 = e_2 = e_3 = e_4 = 0.25, \ q = 0.01\)
## Table: Steady State Probability Distribution

<table>
<thead>
<tr>
<th>Compound</th>
<th>State</th>
<th>Steady State Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\langle 0,0,0,0 \rangle$</td>
<td>0.936254913358677</td>
</tr>
<tr>
<td>1</td>
<td>$\langle 0,0,0,2 \rangle$</td>
<td>0.020767040703947</td>
</tr>
<tr>
<td>1</td>
<td>$\langle 0,0,2,0 \rangle$</td>
<td>0.006443085000445</td>
</tr>
<tr>
<td>1</td>
<td>$\langle 0,2,0,0 \rangle$</td>
<td>0.005896554367512</td>
</tr>
<tr>
<td>1</td>
<td>$\langle 2,0,0,0 \rangle$</td>
<td>0.004721801275921</td>
</tr>
<tr>
<td>2</td>
<td>$\langle 0,0,2,2 \rangle$</td>
<td>0.011734460936930</td>
</tr>
<tr>
<td>2</td>
<td>$\langle 0,2,0,2 \rangle$</td>
<td>0.000103249069863</td>
</tr>
<tr>
<td>2</td>
<td>$\langle 0,2,2,0 \rangle$</td>
<td>0.003596242185866</td>
</tr>
<tr>
<td>2</td>
<td>$\langle 2,0,0,2 \rangle$</td>
<td>0.000101514623954</td>
</tr>
<tr>
<td>2</td>
<td>$\langle 2,0,2,0 \rangle$</td>
<td>0.000028052478081</td>
</tr>
<tr>
<td>2</td>
<td>$\langle 2,2,0,0 \rangle$</td>
<td>0.000028052478081</td>
</tr>
<tr>
<td>3</td>
<td>$\langle 0,2,2,2 \rangle$</td>
<td>0.005204454376759</td>
</tr>
<tr>
<td>3</td>
<td>$\langle 2,0,2,2 \rangle$</td>
<td>0.000049131622044</td>
</tr>
<tr>
<td>3</td>
<td>$\langle 2,2,0,2 \rangle$</td>
<td>0.000042503806239</td>
</tr>
<tr>
<td>3</td>
<td>$\langle 2,2,2,0 \rangle$</td>
<td>0.001243938539611</td>
</tr>
<tr>
<td>4</td>
<td>$\langle 2,2,2,2 \rangle$</td>
<td>0.001401634860264</td>
</tr>
</tbody>
</table>
How to Get There: State Space Analysis

1. compute transition probabilities between each pair of states
   - ⇒ (ergodic) Markov chain
2. compute steady state probability distribution
3. use steady state distribution as initial probability distribution for modified chain
4. transform set of legal states into sink
5. probability mass in set of legal states after $i$ computation steps is $LWA_i$
The Markov Chain Yielding the LWA

<table>
<thead>
<tr>
<th>( \downarrow \text{from/to} \rightarrow )</th>
<th>( \langle 0, 0, 0, 0 \rangle )</th>
<th>( \langle 0, 0, 0, 2 \rangle )</th>
<th>( \langle 0, 2, 0, 0 \rangle )</th>
<th>( \langle 2, 0, 0, 0 \rangle )</th>
<th>( \langle 2, 0, 0, 0 \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle 0, 0, 0, 0 \rangle )</td>
<td>( p(e_1 + e_2 + e_3 + e_4) )</td>
<td>( qe_4 )</td>
<td>( qe_5 )</td>
<td>( qe_2 )</td>
<td>( qe_1 )</td>
</tr>
<tr>
<td>( \langle 0, 0, 0, 2 \rangle )</td>
<td>1</td>
<td>( p(e_1 + e_2 + e_3 + qe_4) )</td>
<td>( p(e_1 + e_2) + qe_3 )</td>
<td>( p(e_1 + e_4) + qe_2 )</td>
<td>( p(e_3 + e_4) + qe_1 )</td>
</tr>
<tr>
<td>( \langle 0, 0, 2, 0 \rangle )</td>
<td>( pe_4 )</td>
<td>( pe_4 )</td>
<td></td>
<td>( pe_3 )</td>
<td></td>
</tr>
<tr>
<td>( \langle 0, 0, 0, 0 \rangle )</td>
<td>( pe_3 )</td>
<td></td>
<td>( pe_2 )</td>
<td>( pe_2 )</td>
<td></td>
</tr>
<tr>
<td>( \langle 0, 2, 0, 0 \rangle )</td>
<td>( pe_2 )</td>
<td></td>
<td>( pe_1 )</td>
<td>( pe_1 )</td>
<td>( pe_1 )</td>
</tr>
<tr>
<td>( \langle 2, 0, 0, 0 \rangle )</td>
<td>( pe_1 )</td>
<td></td>
<td>( pe_1 )</td>
<td></td>
<td>( pe_3 )</td>
</tr>
</tbody>
</table>

**Table:** Transitions Grouped by Number of Operational Processes
Limitations

- computation works for example

- but what about larger systems?
  - state space explosion is obvious

- solution: two ways
  - lumping
  - decomposition
<table>
<thead>
<tr>
<th></th>
<th>Motivation</th>
<th>Basics</th>
<th>Computation of LWAV</th>
<th>Lumpage</th>
<th>Decomposition</th>
<th>Status and Outlook</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Motivation</td>
<td>Basics</td>
<td>Computation of LWAV</td>
<td>Lumpage</td>
<td>Decomposition</td>
<td>Status and Outlook</td>
</tr>
<tr>
<td>2</td>
<td>Basics</td>
<td>Basics</td>
<td>Computation of LWAV</td>
<td>Lumpage</td>
<td>Decomposition</td>
<td>Status and Outlook</td>
</tr>
<tr>
<td>3</td>
<td>Computation of LWAV</td>
<td>Basics</td>
<td>Computation of LWAV</td>
<td>Lumpage</td>
<td>Decomposition</td>
<td>Status and Outlook</td>
</tr>
<tr>
<td>4</td>
<td>Lumpage</td>
<td>Basics</td>
<td>Computation of LWAV</td>
<td>Lumpage</td>
<td>Decomposition</td>
<td>Status and Outlook</td>
</tr>
<tr>
<td>5</td>
<td>Decomposition</td>
<td>Basics</td>
<td>Computation of LWAV</td>
<td>Lumpage</td>
<td>Decomposition</td>
<td>Status and Outlook</td>
</tr>
<tr>
<td>6</td>
<td>Status and Outlook</td>
<td>Basics</td>
<td>Computation of LWAV</td>
<td>Lumpage</td>
<td>Decomposition</td>
<td>Status and Outlook</td>
</tr>
</tbody>
</table>
Markov Chain Abstraction (Lumping)

- goal: smaller Markov chains
Markov Chain Abstraction (Lumping)

- goal: smaller Markov chains

- lumping aggregates states and transitions
Markov Chain Abstraction (Lumping)

- goal: smaller Markov chains

- lumping aggregates states and transitions

- question: what states (and transitions) can be lumped still being the LWAV?
Markov Chain Abstraction (Lumping)

- goal: smaller Markov chains
- lumping aggregates states and transitions
- question: what states (and transitions) can be lumped still being the LWAV?
- answer (for this example): all states that have the same amount of incorrect processes
Lumping Example 1/3
Lumping Example 2/3

Lumping aggregates states and transitions.
Lumping Example 2/3

Lumping aggregates states and transitions.

\[\text{prob}(\overrightarrow{v}, \overrightarrow{w}) = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} p(\overrightarrow{v_i}, \overrightarrow{w_j}) \cdot p(v_i)}{\sum_{i=0}^{n} p(v_i)}\]
Lumping Example 3/3

\[ V_1 \rightarrow V_2 \rightarrow V_3 \]
Lumping Example 3/3

\[ p(\vec{v}, \vec{v}) = \frac{p(\vec{v_1}, \vec{v_1}) \cdot p(\vec{v_1}) + p(\vec{v_1}, \vec{v_2}) \cdot p(\vec{v_1})}{p(\vec{v_1}) + p(\vec{v_2}) + p(\vec{v_3})} \]
Small Example: Result

Limiting Window Availability Vector Gradient
for all compounds

![Graph showing the limiting window availability vector gradient for all compounds.](image-url)
1 Motivation

2 Basics

3 Computation of LWAV

4 Lumping

5 Decomposition

6 Status and Outlook
LWA at Large
LWA at Large

$\pi_1$, $\pi_2$, $\pi_3$, $\pi_4$, $\pi_5$, $\pi_6$, $\pi_7$, $\pi_8$, $\pi_9$, $\pi_{10}$

17496 state Markov chain
Decomposing and Lumping

- lumping aggregates states that have something in common
Decomposing and Lumping

- lumping aggregates states that have something in common
- here: lumping of states that have the same amount of defective processes in common
Decomposing and Lumping

- lumping aggregates states that have something in common
- here: lumping of states that have the same amount of defective processes in common
- decomposition allows the construction of (much) smaller sub-Markov chains
Decomposing and Lumping

- lumping aggregates states that have something in common
- here: lumping of states that have the same amount of defective processes in common
- decomposition allows the construction of (much) smaller sub-Markov chains
- recomposition of smaller lumped Markov chains yields the exact result (80 instead of 17496 states)
Decomposition Scheme

$M$ comprises 17496 states
$M_1$ comprises 24 states
$M_2$ comprises 81 states
$M_3$ comprises 81 states
$M_{\pi_4}$ comprises 3 states
$M_{\pi_7}$ comprises 3 states

$M_1,-$ comprises 8 states
$M_2,-$ comprises 27 states
$M_1,-,red$ comprises 3 states
$M_2,-,red$ comprises 3 states
$M_{3,red}$ comprises 4 states
$M_{red}$ comprises 80 states
LWAV Over lumps
LWAV Over lumps

Probability Mass Distribution over Lumps After 100 Steps

Nils Müllner

Unmasking Fault Tolerance: Masking vs. Non-masking Fault-tolerant Systems
LWAV Over lumps

Probability Mass in Lump $13 = \langle 0, 3, 0 \rangle$, $w = 1000$
LWAV Over lumps

Probability Mass in Lump 65 = <0,0,4>, \( w = 1000 \)
LWAV Over lumps

Probability Mass in Lump $17 = \langle 0,0,1 \rangle$, $w = 1000$
Hierarchical Towards Heterarchical Systems 1/2

- fault propagation **unidirectional**
Hierarchical Towards Heterarchical Systems 1/2

- fault propagation unidirectional
- decomposition easy: no cyclic dependencies
Hierarchical Towards Heterarchical Systems 1/2

- fault propagation unidirectional
- decomposition easy: no cyclic dependencies
- what about any-way propagation
Hierarchical Towards Heterarchical Systems 2/2

- hierarchical self-stabilizing systems demand a hierarchy (order) among the processes. Fault propagation strictly occurs from root towards leafs.
Hierarchical Towards Heterarchical Systems 2/2

- **Hierarchical** self-stabilizing systems demand a hierarchy (order) among the processes. Fault propagation strictly occurs from root towards leaves.

- **Semi-hierarchical** self-stabilizing systems possess the ability to dynamically reassign the role of the root. Switching the root is called an *epoch*. Fault propagation during an epoch is unidirectional.
Hierarchical Towards Heterarchical Systems 2/2

- **Hierarchical** self-stabilizing systems demand a hierarchy (order) among the processes. Fault propagation strictly occurs from root towards leaves.

- **Semi-hierarchical** self-stabilizing systems possess the ability to dynamically reassign the role of the root. Switching the root is called an epoch. Fault propagation during an epoch is unidirectional.

- **Heterarchical** self-stabilizing systems achieve their goal in the absence of any order among the processes. Fault propagation can occur in any direction at any time.
| 1 | Motivation |
| 2 | Basics |
| 3 | Computation of $LWAV$ |
| 4 | Lumping |
| 5 | Decomposition |
| 6 | Status and Outlook |
## Current Focus

- **FINA - 22.-25. March**
  - *LWA*, *LWAV*, and *LWAVG*
  - the computation thereof,
  - basics of lumping
  ⇒ will be presented next month at 7th Int’l Symposium on Frontiers of Systems and Network Applications

- **SSS - 22. April:** system decomposition of hierarchical self-stabilizing systems

- **ICPADS - 24. June:** system decomposition of heterarchical self-stabilizing systems either by iterations, or maybe flow equations...

- writing it up
Unmasking Fault Tolerance

goal: determination of the sweet spot

• be as masking as possible

• with as little effort as possible
Unmasking Fault Tolerance

goal: determination of the sweet spot

- be as masking as possible = maximize degree of masking fault tolerance
- with as little effort as possible = minimize time and space redundancy
Unmasking Fault Tolerance

goal: determination of the sweet spot

- be as masking as possible = maximize degree of masking fault tolerance
- with as little effort as possible = minimize time and space redundancy

⇒ determination of the optimal trade-off thereof
Unmasking Fault Tolerance

goal: determination of the sweet spot
  • be as masking as possible = maximize degree of masking fault tolerance
  • with as little effort as possible = minimize time and space redundancy
⇒ determination of the optimal trade-off thereof
Edsger W. Dijkstra.  
Self-Stabilizing Systems in Spite of Distributed Control.  

Shlomi Dolev.  
Self-Stabilization.  

Stéphane Devismes, Sébastien Tixeuil, and Masafumi Yamashita.  
Weak vs. Self vs. Probabilistic Stabilization.  
Felix C. Gärtner.
Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments.

Sandeep S. Kulkarni and Anish Arora.
Compositional Design of Multitolerant Repetitive Byzantine Agreement.


Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. DRAM Errors in the Wild: A Large-Scale Field Study.
Nils Müllner, Abhishek Dhama, and Oliver Theel.

Nils Müllner, Abhishek Dhama, and Oliver Theel.
Deriving a Good Trade-off Between System Availability and Time Redundancy.
Nils Müllner and Oliver Theel.
The Degree of Masking Fault Tolerance vs. Temporal Redundancy.
Unmasking Fault Tolerance: Masking vs. Non-masking Fault-tolerant Systems

Nils Müllner

nils.muellner@informatik.uni-oldenburg.de
Abteilung Systemsoftware und verteilte Systeme
Department für Informatik
Carl von Ossietzky Universität Oldenburg

February 22, 2011