Communication and Concurrency
Lecture 14

Colin Stirling (cps)

School of Informatics

7th November 2013

Faults

1. Major issue in distributed systems is faults (hardware or software)

Faults

1. Major issue in distributed systems is faults (hardware or software)
2. Strategies for handling faults: fault detection and tolerance

Faults

1. Major issue in distributed systems is faults (hardware or software)
2. Strategies for handling faults: fault detection and tolerance
3. Fault detection: aim is to detect a fault before it causes serious problems

Faults

1. Major issue in distributed systems is faults (hardware or software)
2. Strategies for handling faults: fault detection and tolerance
3. Fault detection: aim is to detect a fault before it causes serious problems
4. Fault tolerance: proper system operation continues in presence of faults

Triple modular redundancy

- Following is from Bruns, “Distributed systems analysis with CCS”, Prentice-Hall, 1997.

Triple modular redundancy

- Following is from Bruns, "Distributed systems analysis with CCS", Prentice-Hall, 1997.
- Redundancy of components: basic technique for fault detection and tolerance
- Consider replacing one component that on input gives an output by

Triple modular redundancy

- Following is from Bruns, "Distributed systems analysis with CCS", Prentice-Hall, 1997.
- Redundancy of components: basic technique for fault detection and tolerance
- Consider replacing one component that on input gives an output by
    1. three copies of the system using a splitter and a voter

Triple modular redundancy

- Following is from Bruns, "Distributed systems analysis with CCS", Prentice-Hall, 1997.
- Redundancy of components: basic technique for fault detection and tolerance
- Consider replacing one component that on input gives an output by
  1. three copies of the system using a splitter and a voter
  2. on input the splitter sends it to each duplicated component

Triple modular redundancy

- Following is from Bruns, "Distributed systems analysis with CCS", Prentice-Hall, 1997.
- Redundancy of components: basic technique for fault detection and tolerance
- Consider replacing one component that on input gives an output by
  1. three copies of the system using a splitter and a voter
  2. on input the splitter sends it to each duplicated component
  3. the voter accepts outputs and outputs majority value

Triple modular redundancy

- Following is from Bruns, "Distributed systems analysis with CCS", Prentice-Hall, 1997.
- Redundancy of components: basic technique for fault detection and tolerance
- Consider replacing one component that on input gives an output by
  1. three copies of the system using a splitter and a voter
  2. on input the splitter sends it to each duplicated component
  3. the voter accepts outputs and outputs majority value
- It works in presence of both "transient" and "permanent" faults

Triple modular redundancy

- Following is from Bruns, “Distributed systems analysis with CCS”, Prentice-Hall, 1997.
- Redundancy of components: basic technique for fault detection and tolerance
- Consider replacing one component that on input gives an output by
  1. three copies of the system using a splitter and a voter
  2. on input the splitter sends it to each duplicated component
  3. the voter accepts outputs and outputs majority value
- It works in presence of both “transient” and “permanent” faults
- Let TMR be triple modular redundancy.

Simple TMR

- Describe a simple TMR system and show that if the number of simultaneous faults is at most one then it behaves the same as a fault-free system

Simple TMR

- Describe a simple TMR system and show that if the number of simultaneous faults is at most one then it behaves the same as a fault-free system
- Agent S receives input at `in` and passes it to the modules M_i, $1 \leq i \leq 3$

Simple TMR

- Describe a simple TMR system and show that if the number of simultaneous faults is at most one then it behaves the same as a fault-free system
- Agent S receives input at `in` and passes it to the modules M_i, $1 \leq i \leq 3$
- Agent M_i receives input at port $\mathtt{mi}_i$ and passes output at $\overline{\mathtt{mo}}$ which may be corrupted

Simple TMR

- Describe a simple TMR system and show that if the number of simultaneous faults is at most one then it behaves the same as a fault-free system
- Agent S receives input at `in` and passes it to the modules M_i, $1 \leq i \leq 3$
- Agent M_i receives input at port $\mathtt{mi}_i$ and passes output at $\overline{\mathtt{mo}}$ which may be corrupted
- Agent V receives outputs at `mo` and passes the majority output value to $\overline{\mathtt{out}}$

Simple TMR

- Describe a simple TMR system and show that if the number of simultaneous faults is at most one then it behaves the same as a fault-free system
- Agent S receives input at `in` and passes it to the modules M_i, $1 \leq i \leq 3$
- Agent M_i receives input at port $\mathtt{mi}_i$ and passes output at $\overline{\mathtt{mo}}$ which may be corrupted
- Agent V receives outputs at `mo` and passes the majority output value to $\overline{\mathtt{out}}$
- Add acknowledgement between V and S

Simple TMR II

$$
\begin{aligned}
S &\stackrel{\text{def}}{=} \mathtt{in}(x).(\overline{\mathtt{mi}}_1(x).(\overline{\mathtt{mi}}_2(x).\overline{\mathtt{mi}}_3(x).S' + \overline{\mathtt{mi}}_3(x).\overline{\mathtt{mi}}_2.S') \\
&\qquad + (\overline{\mathtt{mi}}_2(x)\ldots) \\
&\qquad + (\overline{\mathtt{mi}}_3(x)\ldots)\ldots\ldots\ldots) \\
S' &\stackrel{\text{def}}{=} \mathtt{ok}.S \\
M_i &\stackrel{\text{def}}{=} \mathtt{mi}_i(x).(\overline{\mathtt{mo}}(x).M_i + \textstyle\sum\{\overline{\mathtt{mo}}(v).M_i \ : \ v \in D\}) \\
V &\stackrel{\text{def}}{=} \mathtt{mo}(x_1).\mathtt{mo}(x_2).\mathtt{mo}(x_3). \\
&\quad \textbf{if } x_1 = x_2 \textbf{ then } \overline{\mathtt{out}}(x_1).V' \textbf{ else } \overline{\mathtt{out}}(x_3).V' \\
V' &\stackrel{\text{def}}{=} \overline{\mathtt{ok}}.V
\end{aligned}
$$

$$
\mathtt{TMR}_1 \equiv (S|M_1|M_2|M_3|V)\backslash\{\mathtt{mi}_i, \mathtt{mo}, \mathtt{ok}\}
$$

Simple TMR II

$$
\begin{aligned}
S &\stackrel{\text{def}}{=} \mathtt{in}(x).(\overline{\mathtt{mi}}_1(x).(\overline{\mathtt{mi}}_2(x).\overline{\mathtt{mi}}_3(x).S' + \overline{\mathtt{mi}}_3(x).\overline{\mathtt{mi}}_2.S') \\
&\qquad + \ (\overline{\mathtt{mi}}_2(x)\ldots) \\
&\qquad + \ (\overline{\mathtt{mi}}_3(x)\ldots)\ldots\ldots\ldots) \\
S' &\stackrel{\text{def}}{=} \mathtt{ok}.S \\
M_i &\stackrel{\text{def}}{=} \mathtt{mi}_i(x).(\overline{\mathtt{mo}}(x).M_i + \sum\{\overline{\mathtt{mo}}(v).M_i \ : \ v \in D\}) \\
V &\stackrel{\text{def}}{=} \mathtt{mo}(x_1).\mathtt{mo}(x_2).\mathtt{mo}(x_3). \\
&\quad \textbf{if } x_1 = x_2 \textbf{ then } \overline{\mathtt{out}}(x_1).V' \textbf{ else } \overline{\mathtt{out}}(x_3).V' \\
V' &\stackrel{\text{def}}{=} \overline{\mathtt{ok}}.V
\end{aligned}
$$

$$\mathtt{TMR}_1 \equiv (S|M_1|M_2|M_3|V)\backslash\{\mathtt{mi}_i, \mathtt{mo}, \mathtt{ok}\}$$

Note $\mathtt{TMR}_1 \not\approx \mathtt{Cop}$ Why?

Simple TMR III

- Need to capture that $\texttt{TMR}_1$ behaves like Cop if at most one module produces a fault.

Simple TMR III

- Need to capture that $\texttt{TMR}_1$ behaves like `Cop` if at most one module produces a fault.
- Exercise: How to do this ?

Simple TMR III

- Need to capture that $\texttt{TMR}_1$ behaves like $\texttt{Cop}$ if at most one module produces a fault.
- Exercise: How to do this ?
- Let MP_i, $1 \leq i \leq 3$, be a perfect module.

$$MP_i \stackrel{\text{def}}{=} \texttt{mi}_i(x).\overline{\texttt{mo}}(x).MP_i$$

Simple TMR III

- Need to capture that $\texttt{TMR}_1$ behaves like $\texttt{Cop}$ if at most one module produces a fault.
- Exercise: How to do this ?
- Let MP_i, $1 \leq i \leq 3$, be a perfect module.

$$MP_i \stackrel{\text{def}}{=} \texttt{mi}_i(x).\overline{\texttt{mo}}(x).MP_i$$

- Instead of

$$\texttt{TMR}_1 \equiv (S|M_1|M_2|M_3|V)\backslash\{\texttt{mi}_i, \texttt{mo}, \texttt{ok}\}$$

assume just one faulty module

$$\texttt{TMR}_1' \equiv (S|M_1|MP_2|MP_3|V)\backslash\{\texttt{mi}_i, \texttt{mo}, \texttt{ok}\}$$

Simple TMR III

- Need to capture that $\mathtt{TMR}_1$ behaves like $\mathtt{Cop}$ if at most one module produces a fault.
- Exercise: How to do this ?
- Let MP_i, $1 \leq i \leq 3$, be a perfect module.

$$MP_i \stackrel{\text{def}}{=} \mathtt{mi}_i(x).\overline{\mathtt{mo}}(x).MP_i$$

- Instead of

$$\mathtt{TMR}_1 \equiv (S|M_1|M_2|M_3|V)\backslash\{\mathtt{mi}_i, \mathtt{mo}, \mathtt{ok}\}$$

assume just one faulty module

$$\mathtt{TMR}_1' \equiv (S|M_1|MP_2|MP_3|V)\backslash\{\mathtt{mi}_i, \mathtt{mo}, \mathtt{ok}\}$$

- Now $\mathtt{TMR}_1' \approx \mathtt{Cop}$

Simple TMR III

- Need to capture that $\texttt{TMR}_1$ behaves like $\texttt{Cop}$ if at most one module produces a fault.
- Exercise: How to do this ?
- Let MP_i, $1 \leq i \leq 3$, be a perfect module.

$$MP_i \stackrel{\text{def}}{=} \texttt{mi}_i(x).\overline{\texttt{mo}}(x).MP_i$$

- Instead of

$$\texttt{TMR}_1 \equiv (S|M_1|M_2|M_3|V)\backslash\{\texttt{mi}_i, \texttt{mo}, \texttt{ok}\}$$

assume just one faulty module

$$\texttt{TMR}_1' \equiv (S|M_1|MP_2|MP_3|V)\backslash\{\texttt{mi}_i, \texttt{mo}, \texttt{ok}\}$$

- Now $\texttt{TMR}_1' \approx \texttt{Cop}$
- Exercise: produce the weak bisimulation

TMR with error detection

A more realistic TMR involves error detection.

- the interface includes $\texttt{fault}_i$ and $\texttt{detect}_i$ ports (as well as $\texttt{in}$ and $\overline{\texttt{out}}$)

TMR with error detection

A more realistic TMR involves error detection.

- the interface includes $\texttt{fault}_i$ and $\texttt{detect}_i$ ports (as well as $\texttt{in}$ and $\overline{\texttt{out}}$)
- $\texttt{fault}_i$ models module faults

TMR with error detection

A more realistic TMR involves error detection.

- the interface includes $\texttt{fault}_i$ and $\texttt{detect}_i$ ports (as well as $\texttt{in}$ and $\overline{\texttt{out}}$)
- $\texttt{fault}_i$ models module faults
- to detect faults we add to each basic module a disagreement detector that compares the value computed by the module with the majority value reported by voter.

TMR with error detection

A more realistic TMR involves error detection.

- the interface includes $\texttt{fault}_i$ and $\texttt{detect}_i$ ports (as well as $\texttt{in}$ and $\overline{\texttt{out}}$)
- $\texttt{fault}_i$ models module faults
- to detect faults we add to each basic module a disagreement detector that compares the value computed by the module with the majority value reported by voter.
- Components
  S splitter
  M_i and D_i modules and detectors
  V voter

TMR with error detection II

$$
\begin{array}{lcl}
S & \stackrel{\text{def}}{=} & \mathtt{in}(x).(\overline{\mathtt{mi}}_1(x).(\overline{\mathtt{mi}}_2(x).\overline{\mathtt{mi}}_3(x).\mathtt{ok}.S + \overline{\mathtt{mi}}_3(x).\overline{\mathtt{mi}}_2.\mathtt{ok}.S) \\
 & & \quad + \;\; (\overline{\mathtt{mi}}_2(x)\ldots) + (\overline{\mathtt{mi}}_3(x)\ldots)\ldots\ldots\ldots) \\
M'_i & \stackrel{\text{def}}{=} & \mathtt{mi}_i(x).(\overline{\mathtt{mo}}_i(x).M'_i + \overline{\mathtt{fault}}.\sum\{\overline{\mathtt{mo}}_i(v).M'_i \;:\; v \in D\}) \\
D_i & \stackrel{\text{def}}{=} & \mathtt{mo}_i(x).\overline{\mathtt{do}}(x).D'_i(x) \\
D'_i(x) & \stackrel{\text{def}}{=} & \mathtt{vo}(y).(\textbf{if } x \neq y \textbf{ then } \overline{\mathtt{detect}_i}.D_i \textbf{ else } D_i) \\
V' & \stackrel{\text{def}}{=} & \mathtt{do}(x_1).\mathtt{do}(x_2).\mathtt{do}(x_3).\textbf{if } x_1 = x_2 \textbf{ then } V''(x_1) \textbf{ else } V''(x_3) \\
V''(x) & \stackrel{\text{def}}{=} & \overline{\mathtt{vo}}(x).\overline{\mathtt{vo}}(x).\overline{\mathtt{vo}}(x).\overline{\mathtt{out}}(x).\overline{\mathtt{ok}}.V' \\
 & & \\
\mathtt{TMR}_2 & \equiv & (S|M'_1|D_1|M'_2|D_2|M'_3|D_3|V')\backslash\{\mathtt{mi}_i, \mathtt{do}_i, \mathtt{vo}_i, \mathtt{mo}_i, \mathtt{ok}\}
\end{array}
$$

TMR with error detection III

- What is the relationship between $\texttt{TMR}_1$ and $\texttt{TMR}_2$?

TMR with error detection III

- What is the relationship between $\mathtt{TMR}_1$ and $\mathtt{TMR}_2$?
- Problem $\mathtt{TMR}_2$ has observable actions $\overline{\mathtt{fault}}$ and $\overline{\mathtt{detect}}_i$ (besides $\mathtt{in}$ and $\overline{\mathtt{out}}$)

TMR with error detection III

- What is the relationship between $\texttt{TMR}_1$ and $\texttt{TMR}_2$?
- Problem $\texttt{TMR}_2$ has observable actions $\overline{\texttt{fault}}$ and $\overline{\texttt{detect}}_i$ (besides $\texttt{in}$ and $\overline{\texttt{out}}$)
- How can we “abstract” from them?

Abstracting actions I

- Suppose we have system W that can do actions K and system W' that can do K and the extra action a.

Abstracting actions I

- Suppose we have system W that can do actions K and system W' that can do K and the extra action a.
- We want to relate W and W'. We can abstract from a by "transforming" it into τ.

Abstracting actions I

- Suppose we have system W that can do actions K and system W' that can do K and the extra action a.
- We want to relate W and W'. We can abstract from a by "transforming" it into τ.
- Let $A \stackrel{\text{def}}{=} \overline{a}.A$

Abstracting actions I

- Suppose we have system W that can do actions K and system W' that can do K and the extra action a.
- We want to relate W and W'. We can abstract from a by "transforming" it into τ.
- Let $A \stackrel{\text{def}}{=} \overline{a}.A$
- Let $W'' \equiv (W'|A)\backslash\{a\}$

Abstracting actions I

- Suppose we have system W that can do actions K and system W' that can do K and the extra action a.
- We want to relate W and W'. We can abstract from a by "transforming" it into τ.
- Let $A \stackrel{\mathrm{def}}{=} \overline{a}.A$
- Let $W'' \equiv (W'|A)\backslash\{a\}$
- Now we can ask: $W \approx W''$?

Abstracting actions II

- Abstract from actions

$$\begin{aligned}
\mathtt{Ab} &\stackrel{\text{def}}{=} \mathtt{fault}.\mathtt{Ab} + \sum\{\mathtt{detect}_i.\mathtt{Ab} \,:\, 1 \leq i \leq 3\} \\
\mathtt{TMR}_2' &\equiv (\mathtt{TMR}_2|\mathtt{Ab})\backslash\{\mathtt{fault}, \mathtt{detect}_i\}
\end{aligned}$$

Abstracting actions II

- Abstract from actions

$$\texttt{Ab} \stackrel{\text{def}}{=} \texttt{fault}.\texttt{Ab} + \sum\{\texttt{detect}_i.\texttt{Ab} \;:\; 1 \leq i \leq 3\}$$

$$\texttt{TMR}_2' \equiv (\texttt{TMR}_2|\texttt{Ab})\backslash\{\texttt{fault}, \texttt{detect}_i\}$$

- Exercise Prove that $\texttt{TMR}_1 \approx \texttt{TMR}_2'$

Concurrent systems: alternatives/extensions

- Lots more examples of systems defined in CCS: recent example is web services

Concurrent systems: alternatives/extensions

- Lots more examples of systems defined in CCS: recent example is web services
- Alternatives other process calculi (CSP, ...), petri nets, IO-automata, ...

Concurrent systems: alternatives/extensions

- Lots more examples of systems defined in CCS: recent example is web services
- Alternatives other process calculi (CSP, ...), petri nets, IO-automata, ...
- Maintain basic model of transition systems (vertices as states/processes, edges as transitions)

Concurrent systems: alternatives/extensions

- Lots more examples of systems defined in CCS: recent example is web services
- Alternatives other process calculi (CSP, ...), petri nets, IO-automata, ...
- Maintain basic model of transition systems (vertices as states/processes, edges as transitions)
- Correctness through equivalence and model-checking

Concurrent systems: alternatives/extensions

- Lots more examples of systems defined in CCS: recent example is web services
- Alternatives other process calculi (CSP, ...), petri nets, IO-automata, ...
- Maintain basic model of transition systems (vertices as states/processes, edges as transitions)
- Correctness through equivalence and model-checking
- Extensions pi-calculus (for mobility), adding quantities (time, probability, ...) for modelling embedded/hybrid/biological systems

Concurrent systems: alternatives/extensions

- Lots more examples of systems defined in CCS: recent example is web services
- Alternatives other process calculi (CSP, ...), petri nets, IO-automata, ...
- Maintain basic model of transition systems (vertices as states/processes, edges as transitions)
- Correctness through equivalence and model-checking
- Extensions pi-calculus (for mobility), adding quantities (time, probability, ...) for modelling embedded/hybrid/biological systems
- Requires changes to basic model of transition graphs

Concurrent systems: alternatives/extensions

- Lots more examples of systems defined in CCS: recent example is web services
- Alternatives other process calculi (CSP, ...), petri nets, IO-automata, ...
- Maintain basic model of transition systems (vertices as states/processes, edges as transitions)
- Correctness through equivalence and model-checking
- Extensions pi-calculus (for mobility), adding quantities (time, probability, ...) for modelling embedded/hybrid/biological systems
- Requires changes to basic model of transition graphs
- Correctness is more complex (timed/probabilistic/... bisimulations and temporal logics)

Concurrent systems: alternatives/extensions

- Lots more examples of systems defined in CCS: recent example is web services
- Alternatives other process calculi (CSP, ...), petri nets, IO-automata, ...
- Maintain basic model of transition systems (vertices as states/processes, edges as transitions)
- Correctness through equivalence and model-checking
- Extensions pi-calculus (for mobility), adding quantities (time, probability, ...) for modelling embedded/hybrid/biological systems
- Requires changes to basic model of transition graphs
- Correctness is more complex (timed/probabilistic/... bisimulations and temporal logics)
- Finish course: algorithms for model checking and equivalence checking on finite transition systems