Adaptive Fault Tolerance on ROS: A Component-Based Approach

Jean-Charles Fabre, Michael Lauer, Matthieu Amy

LAAS-CNRS, Ave du Colonel Roche, F-31400 Toulouse, France
Definitions

Dependability: Ability to provide services that can defensibly be trusted within a time-period.

1. Prevention: means to prevent the occurrence or introduction of faults

2. Removal: means to reduce the number and severity of faults

3. Forecasting: means to estimate the present number, the future incidence, and the likely consequences of faults.

4. Tolerance: means to avoid service failures in the presence of faults.
Definitions

**Dependability:** Ability to provide services that can defensibly be trusted within a time-period.

**Fault Tolerance (FT):** Design and implementation of mechanisms to control errors (residual, random, systematic...) by detecting them and ensuring transitions to a safe state.

**Resiliency:** The persistence of dependability when facing changes.

**Adaptive Fault Tolerance (AFT):** Design and implementation of Fault Tolerant Mechanisms (FTM) to ensure the dependability of the system at runtime when facing changes.
Problem statement and key concepts

Once the system is deployed, it faces changes.

System designers cannot predict everything.

Persistence of dependability requires the adaptation of safety mechanism.
Once the system is deployed, it faces changes.

System designers cannot predict everything.

Persistence of dependability requires the adaptation of safety mechanism.

Key concepts for Adaptive Fault Tolerance
- Separation of concerns
- Design for adaptation
- Remote fine-grained updates
Overall process

FTM as a Lego system
Overall process

FTM as a Lego system
Overall process

FTM as a Lego system

Remote update
- Component graph
- Suspend execution
- Modification of the graph
- Re-activate
Overall process

FTM as a Lego system

Change

- Safety analysis / FMECA
- Impact on safety mechanism
- Agile update of FTM
- Remote update

Remote update
- Component graph
- Suspend execution
- Modification of the graph
- Re-activate
### Assumptions and FTM Characteristics

<table>
<thead>
<tr>
<th>Assumptions / FTM</th>
<th>PBR</th>
<th>LFR</th>
<th>TR</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fault Model (FT)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crash</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Transient</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td><strong>Application behaviour (A)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deterministic</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>State access</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Resources (R)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bandwidth</td>
<td>high</td>
<td>low</td>
<td>nil</td>
</tr>
<tr>
<td># CPU</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

PBR=Primary-Backup Replication  
LFR=Leader-Follower Replication  
TR=Time Redundancy
Assumptions and FTM Characteristics

<table>
<thead>
<tr>
<th>Assumptions / FTM</th>
<th>PBR</th>
<th>LFR</th>
<th>TR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault Model (FT)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crash</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Transient</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Application behaviour (A)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deterministic</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>State access</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resources (R)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bandwidth</td>
<td>high</td>
<td>low</td>
<td>nil</td>
</tr>
<tr>
<td># CPU</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

PBR=Primary-Backup Replication
LFR=Leader-Follower Replication
TR=Time Redundancy
Assumptions and FTM Characteristics

**TRANSITIONS**

<table>
<thead>
<tr>
<th>Assumptions / FTM</th>
<th>PBR</th>
<th>LFR</th>
<th>TR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault Model (FT)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crash</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Transient</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Application behaviour (A)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deterministic</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>State access</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resources (R)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bandwidth</td>
<td>high</td>
<td>low</td>
<td>nil</td>
</tr>
<tr>
<td># CPU</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

PBR=Primary-Backup Replication  
LFR=Leader-Follower Replication  
TR=Time Redundancy

Trigger: high rate of HW transient faults observed

Trigger: Non deterministic SW application version

Trigger: bandwidth drop below a given threshold
Componentization of FTM

Change model
Design for adaptation of FTMs
Component-based implementation
Transitions between FTMs

Component-based implementation

Client

Server

request
reply

fault tolerant processing
tag application service

fault tolerant processing

application service

<table>
<thead>
<tr>
<th>FTM</th>
<th>Before</th>
<th>Proceed</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBR (primary)</td>
<td></td>
<td>Compute</td>
<td>Checkpointing</td>
</tr>
<tr>
<td>PBR (backup)</td>
<td></td>
<td></td>
<td>State update</td>
</tr>
<tr>
<td>LFR (leader)</td>
<td>Forward request</td>
<td>Compute</td>
<td>Notify</td>
</tr>
<tr>
<td>LFR (follower)</td>
<td>Handle request</td>
<td>Compute</td>
<td>Handle notification</td>
</tr>
<tr>
<td>TR</td>
<td>Save/restore state</td>
<td>Compute</td>
<td>Compare</td>
</tr>
</tbody>
</table>
Componentization of FTM

Design for adaptation of FTMs

Component-based implementation

Transitions between FTMs

Client

Server

request
reply

Before
Proceed
After

FTM

PBR (primary)

PBR (backup)

LFR (leader)

LFR (follower)

TR

Compute

Forward request

Handle request

Save/restore state

Checkpointing

State update

Notify

Handle notification

Compare

application service
Componentization of FTM

Component-based implementation

Transitions between FTMs

Design for adaptation of FTMs

Change model

Componentization of FTM

<table>
<thead>
<tr>
<th>FTM</th>
<th>Before</th>
<th>Proceed</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBR (primary)</td>
<td></td>
<td>Compute</td>
<td>Checkpointing</td>
</tr>
<tr>
<td>PBR (backup)</td>
<td></td>
<td></td>
<td>State update</td>
</tr>
<tr>
<td>LFR (leader)</td>
<td>Forward request</td>
<td>Compute</td>
<td>Notify</td>
</tr>
<tr>
<td>LFR (follower)</td>
<td>Handle request</td>
<td>Compute</td>
<td>Handle notification</td>
</tr>
<tr>
<td>TR</td>
<td>Save/restore</td>
<td>Compute</td>
<td>Compare</td>
</tr>
</tbody>
</table>
Design for FTM adaptation on ROS

Generic computation graph for FTM
(Boxes represent nodes)

- Topics(0)
- Nodes(2)
  - Client
  - Server

Services: clt2srv (client to server)
Design for FTM adaptation on ROS

Generic computation graph for FTM
(Boxes represent nodes)

- **Topics (6)**
  - pxy2pro
  - pxy2bfr, bfr2prd, prd2aft
  - aft2pro
  - pro2pxy

- **Nodes (5+2)**
  - Client
  - Server
  - Proxy
  - Protocol
  - Before, Proceed, After

**Services:** clt2pxy (client to proxy) and prd2srv (proceed to server)
Implementing PBR on ROS
Implementing PBR on ROS
Implementing PBR on ROS

CLIENT

- Client
- Recovery

PROXY

- clt2pxy

SERVICE

- Service

TOPIC

- Topic

BACK-UP

- Before
- Proceed
- After

PROTOCOL

- pro2bfr
- prd2srv_S
- prd2aft
- aft2aft
- aft2pro
- setstate

SLAVE

- CD_S

CD_S

- cd2rec
Implementing TR on ROS

CLIENT

- Client
- clt2pxy
- pxy2pro
- pro2pxy

Protocol

- pro2bfr
- aft2bfr

TR

- Before
- Proceed
- After

MASTER

- Server_M
- prd2srv_M
- prd2aft
- aft2pro
- aft2bfr
- bfr2prd

Service

- setstate_M
- getstate_M

Topic

- Client
- Proxy
- Protocol
- TR
- Server_M
- Service
- Topic
Combining FTM on ROS

**Generic composition graph for FTM**

- **Protocol node is a software rack of nodes**
  - Before
  - Proceed ➔ activation of services or protocols
  - After

- **Protocol node can substitute for proceed node**
  - It can be view as a frontend of the server…
Combining FTM on ROS

Generic composition graph for FTM

- **Protocol node is a software rack of nodes**
  - Before
  - Proceed ➔ activation of services or protocols
  - After
- **Protocol node can substitute for proceed node**
  - It can be view as a frontend of the server…
• Protocol node is a software rack of nodes
  – Before
  – Proceed ➔ activation of services or protocols
  – After
• Protocol node can substitute for proceed node
  – It can be view as a frontend of the server…
Combining PBR+TR on ROS

Client Recovery

Before After

PROTOCOL

CLIENT

MASTER

TR

Before Proceed After

CD_M

Server_M

getstate_M
setstate_M

TR

Before Proceed After

Server_M

getstate_S
setstate_S

CD_S

PRIOXY

PROTOCOL

BACK-UP

Before After

PROTOCOL

PROTOCOL

PROTOCOL

PROTOCOL

PROTOCOL

PROTOCOL

Before After

Before After

Before After
Case Study

- **Initialization**
  - Initialisation time around **0.5s**
  - Time due to the initialization of communications by the ROS Master

- **Execution**
  - Around **5ms** for the PBR and **2ms** for the TR
  - Requests every 7cm for a car driving at 50km.h⁻¹

- **Recovery**
  - Recovery → Reactivation of **2 Topics**
  - Recovery time around **1ms**

- **Adaptation & Composition**
  - Adaptation → Initialization of new nodes
  - Same order as Initialization time (≈ **0.3s**)

Ubuntu Trusty 14.04
I5 Dual Core 2,5GHz
8Go DDR3 RAM
ROS Master: A single point of failure

- The ROS Master is requisite for:
  - The control over the system
  - The control over communication
  - The control over the graph
  - The control over the Nodes

- If the ROS Master crash:
  - Loss of the software architecture
  - Nodes have to be reloaded
  - The state of the system is reinitialized
  - Critical loss in case of embedded systems

Solutions to assure the reliability of the ROS Master:

- Launching it on a distinct and reliable machine
- Check-pointing its state and restoring it
DMTCP: Check Pointing the ROS Master

• DMTCP, how does it work:
  – Works with Linux kernel 2.6.9 and later
  – Transparent (no recompilation…)
  – Virtualization of Process ID

• Check pointing with DMTCP:
  – Process is launch along the coordinator
  – A checkpoint image is created for each process
  – A restart script is created by coordinator

→ DMTCP should be able to checkpoint the ROS Master

→ The lost of the ROS Master should no longer be a problem
Lessons learnt

• **Adaptive fault tolerance**
  – Separation of Concern
  – Design for Adaptation

  \[ \text{SoC+D4A} \rightarrow \text{FTM isolation and componentization} \]

• **Installation or adaptation of an FTM online**
  – Node can be started and stopped
  – Mapping at initialization

• **Node Management**
  – APIs are not provided by ROS for Node Management
  – User signals and System calls fulfill the missing requirements

• **Implementing dynamic binding**
  – Natural dynamic binding is also not provided by ROS
  – Topics and Services are remapped at the initialization
## Summary of dynamic adaptation

<table>
<thead>
<tr>
<th>SoC</th>
<th>ROS nodes, component mapping to nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>D4A Componentized FT design patterns</td>
<td>Protocol-Before-Proceed-After</td>
</tr>
<tr>
<td>Nodes Mngmnt</td>
<td>Unix system calls and ROS commands</td>
</tr>
<tr>
<td>Dynamic Binding</td>
<td>ROS services, ports, topics</td>
</tr>
<tr>
<td></td>
<td>Additional logic to create ports and topics</td>
</tr>
<tr>
<td>Master CKPT</td>
<td>Check point of the ROS Master</td>
</tr>
<tr>
<td></td>
<td>ROS Master is no longer a Single Point of Failure</td>
</tr>
</tbody>
</table>
Conclusion

• **Now…**
  – Adaptive Fault Tolerance for Resilient Computing is possible on ROS
  – Design and validation of FTM s is always carried out offline
  – If application can be terminated and re-launched : adaptation OK
  – Dynamic adaptation :
    • Extended API for dynamic binding
    • Consistency of reconfiguration?

• **Proceeding…**
  – Experiments on ADAS with *Renault SAS*
  – Evolution of *AUTOSAR* into *Adaptive AUTOSAR*
  – Experimentation on *ROS Master* with *DMTCP*