On the Design of Fault-Tolerance in a Decentralized Software Platform for Power Systems

Purboday Ghosh, Scott Eisele, Abhishek Dubey, Mary Metelko, Istvan Madari, Peter Volgyesi, Gabor Karsai
Institute for Software-Integrated Systems, Vanderbilt University

Supported by DOE ARPA-E under award DE-AR0000666
Outline

- Software for Smart Grid
- RIAPS fundamentals
- Fault management architecture
- Example: Transactive Energy App
- Summary
The Energy Revolution: Big Picture

From centralized to *decentralized* and *distributed* energy systems

**Changing Generation Mix**
- Electric Vehicles
- Transactive Energy

**Decentralization**

*Needs: Distributed ‘grid intelligence’ for*
- *Monitoring + control locally and on multiple levels of abstraction*
- *Transactions among peers*
- *Real-time analytics*
- *Autonomous and resilient operation*
The control picture has not changed

Communication Network

Distribution:
Centralized SCADA system managed by the utility company
The control picture has not changed

Communication Network

Questions:
Q: IS THERE A BETTER WAY TO WRITE SOFTWARE FOR THIS?
A: YES, BUT WE NEED BETTER SOFTWARE INFRASTRUCTURE AND TOOLS.
RIAPS Vision

Showing a transmission system, but it applies to distribution systems, microgrids, etc.
RIAPS Details
The Software Platform

Applications
- Remedial Action Scheme
- Microgrid Management
- State Estimation
- Data Analytics
- Energy Management
- Distributed SCADA

Component Framework
- Component Interactions
  - Component Messaging
- Component Scheduling
  - Event/Time-triggered
- Resource Management
  - Resource Management Service
- Fault Management
  - Fault Management Service
- Logging
  - Logging Service
- Lifecycle Management
  - Initialize, Start, Stop, Checkpoint, Destroy
- Language Run-time
  - C/C++, etc.
- Security
  - Access Control
  - Secure Communications
  - Secure Information Flows
- Persistence
  - Persistence Service

Platform Managers
- Application Manager
  - Application Management and Deployment Service
- Coordination Manager
  - Distributed Coordination Service
- Discovery Manager
  - Broker Service
- Time Manager
  - Time Synchronization Service
- Resource Manager
  - Resource Management Service
- Device Manager
  - Device Interface Service
- Fault Manager
  - Fault Management Service
- Security Manager
  - Security Management Service
- Log Manager
  - Logging Service
- Persistence Manager
  - Persistence Service

OS Kernel

Hardware Platform
- Device Interfaces
  - (Sensors/Actuators/Communications/GPS/...)
- Network Interface(s)
- Storage
Applications consist of ‘actors’: distributed processes deployed on a network, serve as containers for ‘components’. Actors are managed by ‘deployment managers’ and supported by a distributed service discovery system.

Components are (mostly) single-threaded event/time-triggered objects that interact with other components via messages. Several interaction patterns are supported.
RIAPS Platform services

- Deployment: installs and manages the execution of application actors
- Discovery: service registry, distributed on all nodes, uses a distributed hash-table in a peer-to-peer fashion
- Time synchronization: maintains a synchronized time-base across the nodes of the network, uses GPS (or NTP) as time base and IEEE-1588 for clock distribution
- Device interfaces: special components that manages specific I/O devices, isolating device protocol details from the application components (e.g. Modbus on a serial port)
- Control node: special node for managing all RIAPS nodes
RIAPS
Resilience

Definition of ‘Resilience’ from Webster:

- Capable of withstanding shock without permanent deformation or rupture
- Tending to recover from or adjust easily to misfortune or change

Sources of ‘misfortune’:

- Hardware: computing node, communication network,...
- Kernel: internal fault or system call failure,...
- Actor: framework code (including messaging layer)...
- Platform service: service crash, invalid behavior,...
- Application component faults: implementation flaw, resource exhaustion, security violation...
RIAPS
Fault management

- **Assumption**
  - Faults can happen anywhere: application, software framework, hardware, network

- **Goal**
  - RIAPS developers shall be able to develop apps that can recover from faults anywhere in the system.

- **Use case**
  - An application component hosted on a remote host stops permanently, the rest of the application detects this and ‘fails over’ to another, healthy component instead.

- **Principle**
  - The platform provides the *mechanics*, but app-specific behavior *must be* supplied by the app developer

*Benefit:* Complex mechanisms that allow the implementation of resilient apps.
RIAPS
Resource management approach

- Resource: memory, CPU cycles, file space, network bandwidth, (access to) I/O devices
- Goal: to protect the ‘system’ from the over-utilization of resources by faulty (or malevolent) applications
- Use case:
  - Runaway, less important application monopolizes the CPU and prevents critical applications from doing their work
- Solution: model-based quota system, enforced by framework
  - Quota for application file space, CPU, network, and memory + response to quota violation – captured in the application model.
  - Run-time framework sets and enforces the quotas (relying on Linux capabilities)
  - When quota violation is detected, application actor can (1) ignore it, (2) restart, (3) shutdown.
    - Detection happens on the level of actors
    - App developer can provide a ‘quota violation handler’
    - If actor ignores violation, it will be eventually terminated
RIAPS
Resource Models

- Resource requirements fall into 4 categories:
  - CPU requirements: a percentage of CPU time (utilization) over a given interval. If interval is missing, it defaults to 1 sec
    cpu 25% over 10 s;
  - Memory requirement: maximum total memory the actor is expected to use
    mem 512 KB;
  - Storage requirement: maximum file space the actor is expected to allocate on the file storage medium
    space 1024 KB;
  - Network requirements: amount of data expected from and to the component through the network:
    net rate 10 kbps ceil 12 kbps burst 1.2k;
RIAPS
Resource management implementation

- Architecture model specifies resource quotas
- Run-time system enforces quotas
  - Uses Linux mechanisms
- Application component is notified
  - Component can take remedial action
- Deployment manager is notified
  - Manager can terminate application actor

```plaintext
actor LimitActor {
  uses {
    cpu max 10 % over 1;//Hard limit, no 'max' is softlimit
    mem 200 mb;        // Mem limit
    space 10 mb;        // File space limit
    net rate 10 kbps ceil 12 kbps burst 1.2 k; // Netlimits
  }
  ...
}
```
## RIAPS

### Fault management model

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Physical</th>
<th>Description</th>
<th>Detected by</th>
</tr>
</thead>
<tbody>
<tr>
<td>What domain does the fault occur in?</td>
<td>Permanent</td>
<td>Fail-stop (malfunction of node or cluster network)</td>
<td>Hardware watchdog and Platform services</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fail-stop other (malfunction of attached device, sensor, etc.)</td>
<td>Device component</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Network link failure</td>
<td>Fault manager module in deployment manager</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Blabbering idiot (runaway publisher)</td>
<td>Actor/Kernel via network resource limits</td>
</tr>
<tr>
<td>Transient</td>
<td>Temporary network disconnection</td>
<td></td>
<td>Fault manager module in deployment manager (network connection monitor)</td>
</tr>
<tr>
<td></td>
<td>Resource exhaustion</td>
<td></td>
<td>Kernel (using resource limits)</td>
</tr>
<tr>
<td>Cyber</td>
<td>Permanent</td>
<td>Fail-stop (actor stop due to e.g. segmentation faults)</td>
<td>Fault manager, using process connector from kernel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Resource violation</td>
<td>Kernel, using resource limits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deadline or response-time violation</td>
<td>Component framework logic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Component-level anomaly</td>
<td>Component framework logic</td>
</tr>
<tr>
<td>Run-time</td>
<td>Byzantine failures</td>
<td></td>
<td>Not handled</td>
</tr>
</tbody>
</table>

| When does the fault occur?       | Design-time    | Happens while designing the system                                           | Design tools: model-based tools, debuggers                                 |
|                                  |                |                                                                              |                                                                            |
|                                  | Deployment-time| Happens while deploying the system                                           | Deployment manager and RIAPS Controller                                    |
|                                  |                |                                                                              |                                                                            |
|                                  | Human-caused   | Unintentional mistake                                                        | Design tools and fault management architecture                             |

Summary of results from analysis
## RIAPS
### Fault management – Implementation (1)

<table>
<thead>
<tr>
<th>Fault location</th>
<th>Error</th>
<th>Detection</th>
<th>Recovery</th>
<th>Mitigation</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>App flaw</td>
<td>actor termination</td>
<td>deplo detects via netlink socket</td>
<td>(warm) restart actor</td>
<td>call term handler; notify peers</td>
<td>libnl - lmdb as program database exceptions</td>
</tr>
<tr>
<td></td>
<td>unhandled exception</td>
<td>framework catches all exceptions</td>
<td>if repeated, (warm) restart</td>
<td>call component fault handler; notify peers about restart</td>
<td></td>
</tr>
<tr>
<td></td>
<td>resource violation</td>
<td>framework detects</td>
<td>if restarted</td>
<td>call app resource handler notify peers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CPU utilization</td>
<td>soft: cgroups cpu</td>
<td>if repeated, restart</td>
<td>notify actor/ call handler</td>
<td>cgroups</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hard: process monitor</td>
<td>if repeated, restart</td>
<td>notify actor/ call handler</td>
<td>psutil mon + SIGXCPU</td>
</tr>
<tr>
<td></td>
<td>Memory utilization</td>
<td>soft: cgroups memory (low)</td>
<td>terminate, restart</td>
<td>notify actor/ call handler</td>
<td>cgroups + SIGUSR1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hard: cgroups memory (critical)</td>
<td>terminate, restart</td>
<td>call termination handler</td>
<td>cgroups + SIGKILL</td>
</tr>
<tr>
<td></td>
<td>Space utilization</td>
<td>soft: notification via netlink</td>
<td>terminate, restart</td>
<td>notify actor/ call handler</td>
<td>pyroute2 + quota</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hard: notification via netlink</td>
<td>terminate, restart</td>
<td>call termination handler</td>
<td>pyroute2 + quota</td>
</tr>
<tr>
<td></td>
<td>Network utilization</td>
<td>via packet stats</td>
<td>if repeated, (warm) restart</td>
<td>notify actor/ call handler notify peers about restart</td>
<td>nethogs</td>
</tr>
<tr>
<td></td>
<td>Deadline violation</td>
<td>time method calls</td>
<td>if repeated, restart</td>
<td>notify component / call handler</td>
<td>timer on method calls</td>
</tr>
<tr>
<td></td>
<td>app freeze</td>
<td>check for thread stopped</td>
<td>terminate, restart actor</td>
<td>notify component; call cleanup handler; notify peers restart</td>
<td>threads</td>
</tr>
<tr>
<td></td>
<td>app runaway</td>
<td>check for method non-terminating</td>
<td>terminate, restart actor</td>
<td>notify component; call cleanup handler; notify peers about restart</td>
<td>watchdog on method calls</td>
</tr>
</tbody>
</table>
## RIAPS

### Fault management – Implementation (2)

<table>
<thead>
<tr>
<th>Fault location</th>
<th>Error</th>
<th>Detection</th>
<th>Recovery</th>
<th>Mitigation</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIAPS flaw</td>
<td>internal actor exception</td>
<td>framework catches all exception</td>
<td>terminate with error; warm restart</td>
<td>call term handler;</td>
<td>exceptions</td>
</tr>
<tr>
<td></td>
<td>disco stop / exception</td>
<td>deplo detects</td>
<td>deplo (warm) restarts disco</td>
<td>if services OK, upon restart restore local service registrations</td>
<td>libnl + netlink</td>
</tr>
<tr>
<td></td>
<td>deplo stop</td>
<td>systemd detects</td>
<td>restart deplo</td>
<td>(cold) restart disco ; restart local apps</td>
<td>Linux</td>
</tr>
<tr>
<td></td>
<td>deplo loses ctrl contact</td>
<td>deplo detects</td>
<td>NIC down -&gt; wait for NIC up; keep trying</td>
<td></td>
<td>Linux</td>
</tr>
<tr>
<td>System (OS)</td>
<td>service stop</td>
<td>systemd detects</td>
<td>systemd restarts</td>
<td>clean (cold) state</td>
<td>Linux</td>
</tr>
<tr>
<td></td>
<td>kernel panic</td>
<td>kernel watchdog</td>
<td>reboot/restart</td>
<td>deplo restarts last active actors</td>
<td></td>
</tr>
<tr>
<td>External I/O</td>
<td>I/O freeze</td>
<td>device actor detects</td>
<td>reset/start HW; device - specific</td>
<td>inform client component</td>
<td>watchdog on method calls</td>
</tr>
<tr>
<td></td>
<td>I/O fault</td>
<td>device actor detects</td>
<td>reset/start HW; device - specific</td>
<td>log, inform client component</td>
<td>custom check</td>
</tr>
<tr>
<td>HW</td>
<td>CPU HW fault</td>
<td>OS crash</td>
<td>reset/reboot</td>
<td>systemd → deplo</td>
<td>Linux</td>
</tr>
<tr>
<td></td>
<td>Mem fault</td>
<td>OS crash</td>
<td>reboot</td>
<td>systemd → deplo</td>
<td>Linux</td>
</tr>
<tr>
<td></td>
<td>SSD fault</td>
<td>filesystem error</td>
<td>reboot/fsck</td>
<td>systemd → deplo</td>
<td>Linux</td>
</tr>
<tr>
<td>Network</td>
<td>NIC disconnect</td>
<td>NIC down</td>
<td>notify actors/call handler</td>
<td>notify actors/call handler</td>
<td>pyroute2 + libnl</td>
</tr>
<tr>
<td></td>
<td>RIAPS disconnect</td>
<td>framework detects</td>
<td>keep trying to reconnect</td>
<td>notify actors/call handler ; recv ops should err with timeout, to be handled by app</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DDoS</td>
<td>deplo monitors p2p</td>
<td>notify actors/call handler</td>
<td>netfilter + iptables</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>network performance</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RIAPS Fault Management
Interaction patterns – notional example

Two nodes: A and B, with their own Deployment Managers (DeploA, DeploB), discovery service instances (DiscoA, DiscoB), and application actors (ActorA, ActorB). Actors of the same app form a peer-to-peer network, whose members are notified upon membership changes.
Experimental evaluation

- Transactive Energy application:
  - Prosumers ‘trade energy’ with the help of a ‘market’. Results (trades) are recorded in a distributed ledger
Experimental evaluation

- **Experiment 1: Network failure**
  - Node gets disconnected, peers get notified – when node is reconnected peers are notified again

The horizontal axis is the elapsed time in seconds. The values shown from bottom to top are: (a) time to notify peer of disconnect, (b) time to notify peer of reconnect, (c) time for actors on a disconnected node to be notified.
Experimental evaluation

- **Experiment 2: Platform failure**
  - Deployment service fails

The horizontal axis is the elapsed time in seconds. From bottom to top the values are: (a) time to notify local actors, (b) time to terminate actors owned by previous deplo, (c) time to clean up other actors owned by previous deplo, (d) time until actors are fully recovered, (e) is the time until the peers know the node has left, and (f) is when the nodes know the peer has rejoined.
Experimental evaluation

- **Experiment 3: Resource violations**
  - CPU utilization limit, memory limit, disk space limits are exceeded

The horizontal axis is the elapsed time in milliseconds. The values indicate the time interval between detection and invocation of the associated handler for (a) Disk Usage limit, (b) Memory Usage limit and (c) CPU limit.
Summary and conclusions

- RIAPS: A software platform for building distributed real-time embedded apps for the Smart Grid
- Provides a number of services to build resilient apps, including resource and fault management
- Design principles:
  - Resource management: enforce the quotas, but notify app
  - Fault management: detect and, to the extend possible, mitigate faults, but inform the app
- Websites:
  - https://riaps.isis.vanderbilt.edu/ - Project
  - https://github.com/RIAPS - Code base
  - https://riaps.github.io/ - Documents
  - https://www.youtube.com/channel/UCwfT8KeF-8M7GKhHS0muawg - Youtube channel