A Recursively Restartable iRoom

JP Schnapper-Casteras,
George Candea,
Armando Fox

Recursive Restarts

- Failure is certain, especially when using heterogeneous components and services (such as in the iRoom)
- Restarts/reboots
  - Return system (mostly) to well-tested, well-understood start state
  - High confidence way to reclaim stale/leaked resources
  - Easy to understand and use
- Most systems do not tolerate unexpected restarts well
- Solution: make system finely restartable (e.g., iRoom) and apply smart restarts, based on the restart dependencies between components
- A software system is RR if it gracefully tolerate successive restarts at multiple levels

The iRoom

- Stanford Interactive Workspaces
- Weiserian ubiquitous computing space
- Built to research:
  - Multi-device, multi-user applications
  - Multimodal and fluid interaction
  - Integration of wall-sized displays
  - Integrates computing appliances (PDA's, scanners, digital cameras)
- Designed to:
  - Contain reusable system software
  - Integrate legacy off-the-shelf applications (e.g., Microsoft Word)

Failure Detection and Recovery

- Currently: entirely manual
- Problematic
  - Requires intimate knowledge of the architecture
  - Based largely on experience, no well-defined process
  - Long recovery time due to long failure diagnosing time
  - Unknown dependencies among components
    - Lead to unnecessary downtime
    - Can result in lost data
  - Human errors

The Re/Starter

Each node has a service, called the Re/Starter, which starts applications:

1. Spawns the application
2. Stores the application's command line and PID
3. Serves the stored information via a TCP server

Three-Pronged Approach to Failure Detection

A watchdog service listens to multicast information and infers application failure:

1. Applications multicast app-level heartbeats, containing app's global PID and execution status
2. OS-level inferences, based on performance data from Win32 API (e.g., memory/CPU usage)
3. End-to-end tests (e.g., place a tuple in event heap and retrieve it)

Suspected Failure

If the watchdog stops receiving heartbeats:

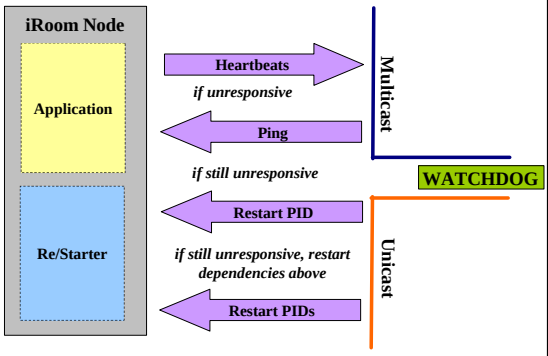
1. Ping the application directly
   (or its wrapper, if third-party application)
2. If no response, issue another end-to-end test, otherwise, back to normal

Recovery

If application end-to-end check fails:

1. Watchdog sends application's global PID to oracle
2. Oracle tells the Re/Starter what to do over TCP/IP
   - Currently, simple oracle requests that the app be restarted
3. If watchdog redetects failure, oracle will
   1. Move toward root of restart map
   2. Infer and add dependencies not captured in restart map
   3. Eventually gives up and requires manual intervention

Failure Detection and Recovery



Platform-Specific Lessons

- Windows problems:
  - Obtaining local PID
  - Obtaining command line
  - Monitoring application liveliness
- Workaround: implement at user level
  - Re/Starter service
  - Three-pronged approach for measuring application-liveliness

Results

- Currently being developed and deployed
- From "unresponsive" to "restarted": on average 3.5 seconds (including network delays)
- Our model and components are useful for:
  - Windows-based services that require high availability
  - Communicating application status over network, as well as logging it
  - Ensuring applications are truly stopped
    - Task manager takes more time
    - "End Task" does not always work

Future Work / Issues

- Scalability
- Reusability in other environment
- Use of virtual machines with suspend/restore to improve efficiency of recovery and reduce MTTR
- Advise applications to checkpoint state before restarting them