Applying Recursive Restartability to Real Systems

George Candea, James Cutler, Armando Fox
& Mercury dev team

Stanford University

Motivation to Reboot

- Reactive restarts: quickly and effectively recover from trouble
  - Deadlock detection in DBMS's
  - NASA Mars Pathfinder
  - How to write production code after 1 week on the job
- Prophylactic restarts: run once for 365 days vs. 365 times for one day
  - Rolling reboots of search engine cluster nodes
  - Patriot missile defense system
  - IBM xSeries servers
- What's good about rebooting?
  - Returns system (mostly) to well-tested, well-understood start state
  - High confidence way to reclaim stale/leaked resources
  - Easy to understand and use
- What's bad about rebooting?
  - Most systems not designed to tolerate unannounced restarts
  - Consequence: long downtimes, potential data loss

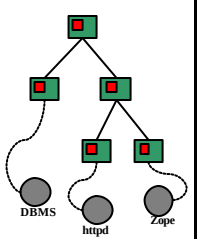
Recursive Restartability

- In ROC systems, must be characterized by
  - Automation, to cut humans out of the loop as much as possible (requires *effectiveness* to justify its existence)
  - Quickness, to reduce MTTR and hence improve availability (requires *minimality* of impact on uptime)
  - Integrity, to not make things worse (requires *simplicity* to build confidence in mechanism)
- Sol: Make systems finely restartable and apply smart restarts
- Definition: A software system is RR if it gracefully tolerate successive restarts at multiple levels
- Key to short/zero MTTRs = partial restarts (both reactive and prophylactic)
- How to build RR systems? Some ideas, still researching.

Core Message

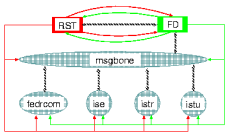
Structuring along MTTF/MTTR boundaries enables the improvement of system availability w/out rewriting code or "rewiring" the infrastructure.

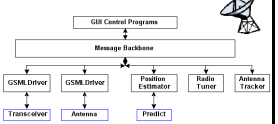
- Restart tree = hierarchy that captures restart dependencies of system (not functional deps, not decision tree)
- Restart group (analogous to UNIX): nodes in a subtree get restarted together
- Strong fault isolation between groups, unequivocal restarting from root (3 trivial + 2 non-trivial r-groups in fig)
- Going up increases MTTR as well as restart confidence → tree hiking policy



Mercury Overview

- COTS-based satellite ground station: Linux+Windows PCs, Java software components, COTS radio modems, etc.
- Failure detection + recovery:
  - Beacons + health summaries → FD detects component failures and reports them to RST [*thorough failure detection was not a goal*]
  - RST: *oracle* tells *restarter* whom to restart
  - FD <-> RST: mutual monitoring and restarting
- Perfect oracle recommends minimal-cost cure policy
- Goal of RR-ification: automate recovery
- Assumptions:
  - All failures are detectable by FD
  - Any component failure will result in temporary unavailability of entire system

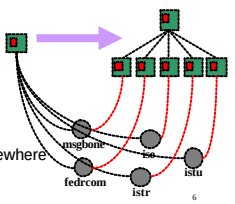




Depth Augmentation

- One restart group → any failure leads to total system restart (actual initial situation)
- Total → partial restart (R_{ise}=2 sec, $R_{fedrcom}$=20 sec)
- Assumptions:
  - Components can restart concurrently
  - Oracle is perfect
  - Restarting does not induce failures elsewhere

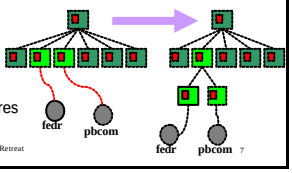
Component	$MTTR_I$	$MTTR_{II}$
ise	289	9.5
istr	289	9.5
istu	289	5.8
fedrcom	289	22.8
msgbone	289	4.7



Subtree Depth Augmentation

- `fedrcom`: high MTTR / low MTTF due to disparate ratio
- Split component along MTTR/MTTF boundaries (we rewired)
- Better MTTR because of fine granularity
- Assumptions:
  - Concurrent restarts
  - Oracle is perfect
  - Restarting does not induce failures elsewhere

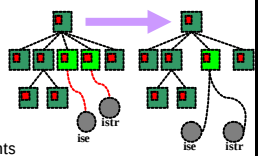
Component	MTTR$_{II}$	MTTR$_{III}$
ise	9.5	9.5
istr	9.5	9.5
istu	5.8	5.8
fedr	22.8	5.0
pbcom	22.8	21.9
msgbone	4.7	4.7



Consolidation

- Cascading failures in ise and istr due to synchronization
- Useless overhead every time resulting from doomed restart
- Encode knowledge in restart tree
- Dual of depth augmentation
- Assumptions:
  - Concurrent restarts
  - Oracle is perfect
- No longer assumed:
  - Independently restartable components

Component	MTTR$_{III}$	MTTR$_{IV}$
ise	28.9	9.5
istr	28.9	9.5
istu	289	5.8
fedr	5.0	5.0
fedrcom	289	228
msgbone	289	4.7

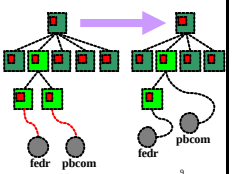


Node Promotion

- Oracle mistakes: guess-too-low and guess-too-high
- Most problematic: widely different MTTRs (`fedr`: 2 sec, `pbcom`: 19 sec)
- Push high-MTTR up, low-MTTR down
- Side effect: free `fedr` rejuvenation
- Assumptions:
  - Concurrent restarts
- No longer assumed:
  - Oracle is perfect
  - Independently restartable components

Faulty oracle (20% mistakes)

Component	MTTR$_{IV}$	MTTR$_{V}$
ise	6.1	6.1
ilstr	6.1	6.1
istu	5.8	5.8
fedr	9.8	9.8
pbcom	26.7	23.9
msgbone	4.7	4.7



Lessons and Discussion

- MTTF/MTTR-based boundary (re)definition instead of "traditional" ways (e.g., `fedrcom` → `fedr` + `pbcom`)
- Transform restart tree post deployment (addresses most "expensive" time to fail in product's life -- the later you discover a bug, the more expensive it is)
- Not all downtime is the same (e.g., satellite pass)... would you rather have high MTTF or low MTTR ?
- Need knowledge of distribution to use MTTF/MTTR in making predictions (typically low coeff of variation assumed)
- Restart group boundaries should not intersect existing failure isolation boundaries

Ongoing Work

- Collect more precise numbers
- Apply RR to Interactive Workspaces Room
- Improve fault detection and logging in ground station
- Design a "RR object" to be inherited by all sw components in system (e.g., a RR EJB in a J2EE-compliant application server)

More...

http://RR.stanford.edu