OSDI Work-in-Progress Abstract, San Diego, CA, October 2000

Reboot-Based High Availability

Turning evil reboots into reliable friends

George Candea, Armando Fox
Stanford University
{candea,fox}@cs.stanford.edu

Large scale, critical software infrastructures consist of industrial strength components that have undergone extensive testing and debugging. Despite this, failures still occur, primarily due to Heisenbugs that are typically "resolved" by rebooting. Furthermore, since scheduled downtime is cheaper than unscheduled downtime, many 24x7 services undergo periodic, prophylactic reboots to avoid failures that may result from software aging (e.g., slow memory leaks). Conceding that Heisenbugs will remain a fact of life, we propose a systematic investigation of reboots as a high-availability mechanism. Our approach is based on:

- "Partial reboots" that minimize time-to-restart;
- An execution infrastructure;
- A set of techniques for making software amenable to restart-based failure management.

A partial reboot is one in which only components that require cleanup are restarted, without bringing the entire service down. At its core lies a "restartability tree"; parent software components provide mechanisms for isolating their children from each other, and can cleanly restart children after reclaiming their resources. For example, processes can be cleanly restarted by the OS, which also provides isolation mechanisms such as virtual memory and I/O multiplexing; a similar analogy applies to threads in a process, though they enjoy weaker isolation from each other.

The execution infrastructure relies on each component providing two methods: PROBE and RESTART. PROBE is called periodically and performs an application-level progress check whose semantics are necessarily application-specific. PROBEs are supplemented by end-to-end checks, such as verifying the response to a well-known query. RESTART, called when PROBE reveals a possible anomaly, advises the component that it should clean up any pending state because it is about to be restarted by its immediate ancestor in the restartability tree. An analogy would be UNIX services that understand the common idiom "kill -TERM; sleep 5; kill -9". If restarting does not eliminate the anomaly, a reboot at a higher level of the hierarchy is attempted (e.g., if the TCP stack is corrupted and restarting the process does not help, rebooting the OS may allow the service to resume).

We also seek to codify rules for (re)structuring applications into components whose interactions are likely to support this simple, effective form of maintaining availability. We are systematically extracting guidelines from existing work on soft state, announce/listen protocols, orthogonal application mechanisms, and loosely coupled distributed systems. The result should make our system well suited to both reactive failure management and proactive software rejuvenation.