15441 Paper Summaries, November 29, 2001
Suman Kumar Nath(sknath@cs.cmu.edu)

Hive: Fault Containment in Shared-Memory Multiprocessors

Main Points:

- The paper describes the design of an operating system for large-scale SMP's with the goal that faults in one small part of the system do not bring the entire system down.
- **Challenge:** Fault containment in distributed shared-nothing architectures is relatively easy. Hive's challenge lies in the fact that the whole system must provide the traditional *single system* image, providing flexible sharing of processors, memory and other resources. Of great concern is the *wild writes* problem where a faulty node can corrupt other nodes' memory.
- **Solution:** Hive employs both hardware and software mechanisms for fault containment. They were fortunate in that their hardware platform, FLASH, was being designed in conjunction with their OS.
- Hive is structured as a set of cells. Each cell is assigned a fixed set of processors (alongwith its memory and I/O devices). Each cell manages its resources, essentially as a (semi-)independent multiprocessor OS. A cell is the smallest unit of the system that will be lost in case of a hardware/software failure. However, individual cells are not visible to the end user. The cells cooperate to provide the required single-system view.
- Different cells communicate via RPC. The kernels never *write* directly into each others' memory. Remote *reads* are allowed. However, at user-level, for efficiency reasons, it is desirable that processors be able to both *read* and *write* into remote memory.
- Hive withstands Operating System failures as follows:
  - *Corrupt Messages:* RPC sanity checks suffice.
  - *Remote reads:* It is the reading cell's responsibility to defend itself against reading corrupt memory. A *careful reference* protocol (an elaborate procedure for reading memory and performing consistency checks on read values) is employed.
  - *Remote writes:* Kernels never write into each other's memory. Such *wild writes* during a kernel crash are caught by a hardware *firewall*.
- **Failure detection and recovery:** All cells monitor themselves at regular intervals using heuristic checks. A failed check triggers recovery immediately. All processors save state and then run a distributed consensus protocol to identify the failed cell(s) and issue their reboot. Failures are also triggered by failed RPC calls, hardware error detection and a failed *careful read* call.

Flaws:

- Applying HIVE approach to an existing off-the-shelf OS would require considerable redesign and reimplementation effort.
- Cells have to perform detailed checks on every message they receive. This may add significant overhead.
- HIVE requires hardware support for efficient fault containment. Commercial SMPs like SGI Origin 2000 do not provide hardware fault containment.