Hypervisor-based Fault Tolerance

This paper is an interesting detour from much of the previous reading we've done in the course. The point of virtualization in this system is not multiplexing of a hardware node. Instead, the goal is to insert a thin hypervisor between the hardware and the operating system in order to coordinate the activities of two or more processors in a fault-tolerant system.

In some ways, this paper reminded me of the Denali paper. In this system, as in Denali, the application for virtualization is quite narrow and specific. The goal here seems to be to provide a general-purpose fault-tolerant system. Presumably, any number of applications of different types could be run on such a system, and the operators/users could be assured that all replicas would maintain the same state, within the capabilities of the system. Also like Denali, the hypervisor is quite small, consisting of some 24,000 lines of code. That 5,000 of these lines are assembly code seems a little daunting. Still, points made early in the paper seem quite valid. Inserting a hypervisor into the system seems like a much more reasonable effort than requiring changes to the hardware, the operating system, or individual applications to achieve the same results. This idea is familiar from other papers we've read.

On some levels, the challenge of coordinating the work of two or more replica processors seems quite straightforward, at least at first. Using the state machine idea, there are two types of instructions: deterministic and non-deterministic. Logically enough, the non-deterministic instructions are those that bring outside input into the system, so-called environment instructions. As we read on, we discover that, as always, the devil is in the details of the implementation.

Yet another familiar idea is that of the processors innate unsuitability for virtualization. Based on our discussions, it seems that some work is being done to equip processors for this task, no doubt in response to all the research being done in virtualization and the growing popularity of this technology.
On the other hand, there are some capabilities in the HP PA-RISC processor that make virtualization feasible. One of these is the recovery register, which enables the hypervisor to gain control of the processor at regular intervals, which define the epochs. Another is the existence of four privilege modes, of which only two are used by the operating system. This is another familiar idea. As in other systems, the hypervisor runs in privilege mode 0, while the operating system runs its privileged instructions in privilege mode 1, and user mode instructions run in privilege mode 3.

I found the architecture of this system interesting. Instead of actually running “below” the OS, the hypervisor appears to the OS as a device driver. This seems like a neat solution and perfectly reasonable, given that there’s a relatively small subset of instructions—the environment instructions—that must be mediated by the hypervisor. One replica designated as the primary, which runs ahead of other replicas and defines what the results of environment instructions will be for the system. The recovery register is used to allow the hypervisors to regain control of the replica systems at the end of each epoch in order to synchronize the states of all the machines. That is, the results of each environment instruction will have identical effect on the state of each replica as of the end of each epoch, and processing then continues.

The overall goal of the system is, of course, fault tolerance. That is, any replica, including the primary, can fail quietly and processing will continue without interruption. The key is that the failure can be handled without incorrect or disruptive behavior and in a way that can be noticed by the system and handled appropriately. That backup replicas lag by one or more epochs behind the primary helps to make this possible. The architecture also designates that in case the primary fails, one of the backups is promoted, but only at the end of the epoch during which the primary fails. This is a trade-off that may result in some data loss, with the advantage that this model makes it possible to support a wider range of IO devices than if a different approach were taken, to promote a backup as of the beginning of the epoch during which primary failure occurred.

The implementation in the prototype system seems quite reasonable. Due to memory and
privilege-level constraints in the processor, only one virtual machine can be supported per hardware
node by the hypervisor. The goal of this system is not multiplexing of hardware, so this hardly seems
like a problem. It was interesting to note that the researchers, as well as the HP engineers, discovered
that not all ordinary instructions on the processor are deterministic, which required some alterations to
the hypervisor to take over TLB management from the operating system. There's also some discussion
of optimizations made to the protocols to streamline communications between the primary and backup
replicas.

In the discussion of the prototype's performance, I was impressed with how accurately the
performance models predicted normalized performance as a function of epoch length, with longer
ePOCHS generally improving performance because, as the article states, “epoch boundary processing is
the dominant cost.”

Overall, I found the normalized performance to be quite reasonable. Even if processing on such
a system is reduced by as much as one half with an untuned hypervisor, the goals of the system would
appear to have been met. As the authors acknowledge, much work remains and there are many
unanswered questions.