CSE 231—Advanced Operating Systems
"ReVirt"
or "What can we do with virtualization??"

Andrew Quinn

A note on the Authors. The advisor on the Revirt paper has been accused of heinous crimes. Incidentally, he was my advisor when he was arrested. I have no idea if he did it, nor is it even reasonable for me to postulate.

This puts this paper in a tough spot. Revit is one of my favorite papers and was instrumental in my thesis. While Revirt isn't the first paper on record and replay, it is the genesis of tons of work on record and replay in the OS, SE, PL, and architecture communities (and, won a test of time award as a result).

Ultimately, I decided to review this paper mainly because academics do not make money from us studying their work; we gain knowledge while they gain very little. Moreover, I decided to cover the work with two main stipulations. First, while we often discuss the people behind the research, we won't do that here. Second, publicity, citations, and review are not equivalent to and endorsement of the people who made the work (as stated above, I have no idea of the authors guilt or innocence).

Summary. The trickiest technical bit in this paper is the discussion for how they handle interrupts. The challenge is that the system needs to ensure that each interrupt is delivered at precisely the same time during replay as it was during recording. Some system prevent interrupts from being raised at arbitrary points in the execution [**?**, **?**]. However, ReVirt instead guarantees that each interrupt will be raised before the same instruction.

The strawman approach instruments the program and counts all executed instructions during recording and replay. Unfortunately, this would be painfully slow. Note, however, that there are other counting schemes that

can still uniquely identify each execution instruction. For example, each pair of (branch count (`bc`), program counter (`pc`)) uniquely defines an executed instruction, so an interrupt raised before the same (bc, pc) pair during recording and replay will be correct.

ReVirt uses to Intel's performance counters to accelerate the pausing before a branch counter. The system tells the performance counters to raise an interrupt after $\texttt{bc} - 1$ branches and then the processor single steps to `pc`. Unfortuantely, performance counters have "skid"—they may raise an execution a few instructions after the `bc`'s branch. So, ReVirt actually chooses an arbitrary value that will always be larger than the skew, and raises the interrupt after $\texttt{bc} - 128$ instructions and then single-steps to the right spot.

Discussion. A major challenge facing record and replay systems is how they will handle concurrency. In particular, how can you handle record and replay of data races? Actually recording and replaying races will be overly expensive, so what can you do?

1. SMP-Revirt [**?**]: use page-protections!
2. DeLorean [**?**]: use hardware!
3. ODR [**?**]/PRES [**?**]: treat replay as search!
4. Arnold [**?**]/Castor [**?**]: Races are bugs! No need to support!

Why might we record an execution? There have been a few major reasons and techniques explored in the literature today:

1. Debugging [**?**]
2. Fault Tolerance [**?**]
3. Online analysis [**?**, **?**]
4. Intrusion Analysis [**?**]
5. Workload Capture [**?**]

Record and replay is a super important approach in the research community for many of these purposes.