Review of paper

Debugging operating systems with time-traveling virtual machines

by Samuel T. King, George W. Dunlap, and Peter M. Chen

1 Summary

This paper takes an innovative approach to debugging operating systems. A para-virtualised OS (user-mode Linux, UML) is run on a hypervisor (Linux with UML support) to enable it being debugged, including arbitrarily stepping forwards and backwards in time. Using simulators/emulators/VMMs for OS debugging is, of course, an established technique. However, the authors approach makes a qualitative difference by very clever use of virtualisation. The approach is also applicable to multi-threaded user code. The paper is very well written, the approach is very comprehensive and thoroughly explained. The authors did am amazing amount of work, even though they leveraged some of their earlier work.

2 Discussion of approach

The work combines a number of techniques and tricks to keep enough state to be able to reproduce any previous state of the system very quickly, and does so with surprisingly little memory and space overhead.

Previous state is kept by (unsurprisingly) performing regular incremental checkpoints. Copy-on-write techniques are employed to keep the checkpoints manageable in size, and disk state is kept by remapping blocks and keeping the old data. Each checkpoint contains a replay log (for going forward) and an undo log (for going backward).

Old checkpoints can be deleted (automatically or manually) to save space, and new ones can be inserted post-facto in order to reduce the cost of rebuilding intermediate state.

The approach requires logging of all external events, such as interrupts and I/O. Events are "timestamped" using branch counts from the x86 performance monitors and the program counter value at the time of the event.

One of the really impressive achievements is the ability to debug device drivers and deal with interrupts and DMA. In general this requires guest execution of real device drivers. This is achieved by virtualising all I/O instructions as well as accesses to DMA buffers. (The latter is helped by drivers using x86 string instructions to access buffer data, which keeps virtualisation overheads low.)

3 Evaluation

Work of this kind is very difficult to evaluate quantitatively. Yet the authors manage to do a very good job in the evaluation.

They demonstrate that the overheads of running the system with checkpointing enabled are impressively low, mostly in the single-digit percentage range, and the space cost seem very bearable as well. This is, of course, a prerequisite to enabling debugging in situations where a bug only manifests itself after days of operation, a situation not untypical for kernel bugs. The smart design that allows later deletion and insertion of checkpoints helps here. Nevertheless, it is not clear whether the overheads would be low enough to debug, say, a Gigabit NIC driver.

I was mostly impressed with the authors coming up with not just one, but four convincingly realistic case studies, a nice change from contrived cases or highly dubious fault injection experiments.

The authors claim that of 98% of OS code is debuggable in spite of para-virtualisation. The paper makes this claim credible, although it would be hard to confirm quantitatively.

4 Shortcomings

I didn't really find anything wrong with this paper. The (minor) criticisms are

- it would have been helpful to be given more of an explanation of what skas does;
- given that there is some dependence on x86 specifics, a discussion of how things might work on other architectures would be appreciated;
- the overhead figures quoted are somewhat misleading as they compare against VM execution without checkpointing. The real comparison is against native execution (although the VM overhead is, of course, dependent on the VMM, and thus a separate issue);
- the authors should indicate how multiprocessor systems could be supported, as otherwise the approach is of much reduced value.

5 Overall impression

This is excellent work, which is likely to have very significant practical use, although its practical use is limited due to the lack of multiprocessor support, and the paper offers no hope that this is forthcoming.