General:
1. Can this be done at a (much) smaller scale to more effectively debug applications?
2. In section 3.2, and the related work, the authors mentioned an approach of using software emulator for the real device in VMM. The authors chose not to use this approach because (1) it requires to have an accurate software emulator, and (2) it is slow. Is there any benefit of using a software emulator although it might be slow?
3. In addition, VMM in this paper forwarded instructions to direct real I/O (e.g., write to disk). Does it pose any potential problem of corrupting the disk? (It seems that the protection using virtual disk has been weakened.)
4. Can the system present here deal with bugs that are sensitive to the timing of instruction executions? Would the logging overhead have an impact on some of these bugs (i.e. maybe hide them)?
5. Are there any additional problems associated with applying the techniques presented in this paper to a fully-virtualized VMM? What adjustments, if any, would need to be made?
6. Is it possible to create a reverse debugger without using a virtual machine? Are there any other techniques that could be used to achieve the same results?
7. This isn't really a question about TTVM per se, but: They say that "ReVirt uses a performance counter on the Intel Pentium 4 CPU..." (p.4). However, ReVirt was actually implemented on an AMD Athlon. The way the TTVM paper words it, though, it sounds like the performance counter is a feature of the P4 alone (rather than of the x86 architecture). I notice that TTVM was partially funded by Intel. Do you think it's appropriate for research being published by universities to skew reality to support funding bodies? Shouldn't the research be done either in-house (Intel does it's own research and releases it's own papers) or externally (Intel may provide funding, but ultimately shouldn't be able to affect the results)? If funding bodies can influence papers like this, how can we actually trust any externally funded research?
8. With their experience from the USB device driver, if the error occurs after a long time, isn't memory an issue? The memory that keeps track of all the interrupt calls that caused the problem. There may be a series of events that together cause the problem, so we do need to keep record all of the events.

Multiprocessors/Extensibility:
1. Like ReVirt, this solution's major hindrance is the lack of multiprocessor support. Since this paper was published within the last 2 years, it's a shame the authors didn't engineer a solution that would apply to current generation hardware; they didn't even list it as potential future work but did make a brief comment on page 4. Is it just too onerous to track the fine-grained interleaving order of memory operations of these multiprocessor architectures?
2. The authors say that "researchers are investigating ways to support replay on multiprocessors." (p4) With today's computers commonly having multiple processors, it is completely unimportant if it can't support them. What advances have been made to support multiple processors?
3. A big problem to debuggers for multi-processor/multi-thread system is the time stamp of the application. In another word, it is very difficult to replay what happened at the bugging point. The little difference of running on different PE or the reduce power schedule will
change the situation of application running. Did TTVM solve this problem?

4. Is this technique scalable to a distributed system?

**Guest OS / Host OS Virtual machine model:**

1. Does the fact that the OS being debugged is running on top of a virtual machine and another OS affect the bugs and debugging?
2. The authors claim that 98% of the host OS code base can be debugged in the guest OS. How did they come up with this number? What kind of code is the last 2%, and why can it not be debugged?
3. The authors state that "applying the techniques in this paper to a non para-virtualized VMM such as VMware would enable reverse debugging to work for any host OS bug." Why and how? Would "any host OS bug" include timing-dependent bugs? Are fully virtualized VMMs likely to be more similar to the OS that runs on the hardware than a para-virtualized VMM (a possible reason)? Does VMware provide reverse debugging?
4. In terms of reverse debugging, what are the pros/cons of adopting a guest OS like UML that creates a separate host process for guest processes compared to UMLinux that creates a single host process that encompasses both guest OS and guest processes?

**Feasibility:**

1. Running a debugger on a virtual machine below the OS seems like a great idea and this paper models it well. Has this been implemented? Would this be used only for developers while trying to debug their OS or could it be extended to run on every machine where it would automatically pick up bugs and send the logs to the developers to be debugged (similar to Microsoft’s error reporting but much more in depth)? Could this be extended even further to automatically pick up bugs and automatically learn to fix common ones?
2. They keep stating that they found their debugger easy to use etc. but they're the ones who created the tool! (Of course they would find it intuitive!) Do people use this TTVM debugger? Do others exist?
3. Ok, it's been mentioned many times that debugging operating systems is a very tough thing to do (and hurray for printf). This paper was released in 2005... so... what are the limitations of this method? (it requires the use of a VM?)
4. This paper is fairly new (2005), how did people conduct OS debugging before using VMs? What were the most effective and competing methods?

**Performance:**

1. It is never mentioned what is a typical period of time their technique is capable of logging, could this be due poor results?
2. It is mentioned that the space overhead of iterative checkpointing is reasonable (2-6 MB/sec at 25 seconds checkpoints, and about 4-7 MB/sec at 10 seconds). What is unreasonable? I thought these numbers are huge, am I missing something? (Maybe the Halloween Party J)
3. They say the space overhead for time travel is low, but if we take an average case of 40KB/sec and run it for a day, it's already at 3.3GB. Do that for a year and it's 1.2TB (worst-case is ~2.5TB)! Is it just me or does this seem like a high space overhead?