

Keith Douglas
Tax Data Division
Statistics Canada¹
kd@prime.gushi.org

Title: "Through an Event Log, Darkly"

Abstract: In this paper I discuss how computer virtualization technologies (e.g. VMWare) provoke metaphysical and epistemological problems (e.g. a version of the the experimenter's regress) similar to but not identical to familiar philosophical concerns. A sketch at resolution of one is proposed; the others are suggested as interesting open research problems in the area of potential limitations of computing.

¹ Institutional affiliation is for identification purposes only. My work in the paper which follows was inspired by my experience with this organization but is not a product of my employment there.

“Through an Event Log, Darkly”

Introduction

Recent work in the philosophy of science has included work (e.g., Lehtinen and Kuorikoski 2007 and the literature cited therein) on the nature of simulation. The philosophy of AI literature (e.g. that found in Hofstadter and Dennett 1981) also debates whether cognition is implementable, simulatable (or not even this). However, an area which has not been sufficiently explored philosophically is the simulation of computers by computers. In the following, I discuss recent developments in simulation of computers by computers in the light of a few philosophical perspectives. The paper is intended as a summary of possible areas of interest and is not an exhaustive study of any of them in particular. The following is thus divided into four sections. In the first, I briefly stage-set by discussing current simulation environments and what they are used for. The following two sections discuss three puzzles of philosophical interest raised by the software systems described in section 1. Finally, I conclude with a call for further research in this area.

Section 1: Simulation of Computers by Computers?

Simulation of computers by computers is not actually new. There have been shareware and commercial emulators (as they are often called) for decades, some of which make use of additional hardware (PC Transporter, the Apple IIe card for the first few LC Macintoshes), some relying more purely on software (Bernie][The Rescue, OS/2's DOS emulation). The more sophisticated of the latter actually made use of special hardware at the CPU level to support software which is explicitly designed to simulate

other systems. This “virtualization” (as it is often called to distinguish it from other forms of simulation) is the technical reason behind IBM’s (failed) marketing campaign in the mid 1990s to sell OS/2 DOS emulation as “better DOS than DOS”. In all of these traditional cases the uses of emulation were one or more of the following: (a) to run legacy software in a new system (e.g. OS/2’s DOS emulation), (b) to run software from a competing platform (e.g. SoftPC/SoftWindows on NeXTStep and MacOS) or (c) to have fun with old stuff - the numerous software emulators of Apple IIs, Commodore 64s, NESes, etc. that are popular to this day.)

What has changed in the past few years is the ability of microprocessors to support virtualization at approximately the same feature parity. The “Virtual 8086” mode which OS/2 made use of was a virtualization of hardware a few revisions behind - hence one could not, for example, run OS/2 itself in the virtualization. Now, with modern microprocessors, the virtualization of modern operating systems is indeed possible. For example, running Windows 2003 Server is completely reasonable in the sense that it is not hopelessly poor performing- for the most part. (The “most part” qualifications are of course the subject of the remaining sections of this paper.) Extremely powerful machines are used to virtualize many powerful “guest” machines as they are sometimes called. This is done now for several reasons: security (add additional layers between the real operating system, hardware, etc. and the user), to recapture lost performance (why buy 10 machines operating at 10% efficiency when you can in principle have 1 big machine at 100% allocation of resources for less than 10 times the cost of a small machine), ease of maintenance, etc. A popular and powerful software package family

that does this is put out by VMWare. I will be taking these products as the basis for my examples, though they apply in general to any such product.

Virtualization does sound fine and dandy and a wonderful way to accomplish the goals I just discussed. But there are interesting problems that result from the virtualization. I turn to the first of these.

Section 2: Simulating a Timer, or why Turing equivalence isn't enough

Turing (1964 [1936]) proved that a number of understandings of calculation are coextensive in the sense that they permit the calculation of the same functions. This in a way is the insight behind virtualization, as a logician would see it. For better or for worse, however, Turing's proof idealizes away notions of time: his result holds provided we are allowed to calculate for any arbitrary finite period as we wish. This is fine for the purpose he had in mind. It is also correct if one simply wants to eventually get an output of a result. But, virtualized hardware is intended to be used like real hardware. To take a concrete example, one can install any software - take Microsoft BizTalk Server - on a virtual machine.

I use this example as it is a timing sensitive piece of software. How do real computers measure time? By counting "ticks" of an oscillator of a specific frequency. But in a virtual machine, this is reliant on the oscillator on the underlying host. Moreover, since the host can only allocate resources to a certain number of virtual machines at a time (potentially as few as one - but never equal to the number of hosts in general or one wouldn't gain many of the benefits of virtualization) these updates do not **and cannot** occur at the time they should. Hence, there is necessarily a mismatch in timing between what the

“real world” expects and what goes on inside a virtual machine (or, alternatively, between two or more virtual machines).

When does this come into play? When timing in the application is important. An obvious example is network latency. The ubiquitous ping is used to measure this property² of networks (and servers, etc.). For example, the current latency between my home machine and the mail server I use is on the order of 19 ms. This measures the round trip duration between my mail server (run by a colleague of mine in New York, so fairly “far away”) and my home computer. Under ordinary circumstances, ping is sufficiently nonintrusive that it is not affected by server or router load along the way and similar considerations.

Ping by itself, however, is not terribly relevant. A lot of places use it merely to verify that a host indeed is still connected and not so overloaded it fails to answer. However, since it does report latency, it behaves (in principle) similar to more complicated tests of connectivity. One such test is used routinely by what Microsoft calls “database mirroring” in SQL Server 2005. In this set up, one server (the **mirror**) receive network copies of the database changes performed on another database server called the **principal**. This setup can be rendered more robust in the case of server failure by making use of a another server whose task is not to receive copies of the data but to insure that the principal is “visible” at all times and to inform the mirror to “fail over”, i.e., become the principal. (If then later the old principal reappears it becomes mirror and the situation is

² Strictly speaking, latency is one way duration-of-transit and so ping measures more or less twice this amount. I shall ignore this subtlety in what follows.

exactly reversed.) This additional server is called the witness. It might be thought that because the witness has no task other than to monitor the other servers its resource usage would be minimal and so it would be an ideal candidate to be virtualized.

This inference is mistaken. In order for the witness to do its job, it must perform a sort of “complicated ping” to the principal and mirror. Suppose then that the timing information on the witness is “out of wack” in the way I described previously. What can happen? The witness can “lose track” of the servers it is supposed to be monitoring because durations (in essence) are either very high or are timed out - and so the witness performs the fail over. This can happen repeatedly and other components to the application making use of the database in question get, one might say, hopelessly confused. In the case of some applications, the failover of the database requires human intervention for further parts of the application in question to work correctly - hence this sort of mishap is unfortunate (to say the least) for those maintaining it. Notice that this has **nothing** to do with resource limitations - we are not trying to simulate a modern 64 bit quad core CPU on an 8 bit CPU.

What, then, is the philosophical lesson of this parable told by a database administrator? It is not merely the old lesson that a simulation is not always identical in a relevant respect to the thing simulated. It is interesting because it illustrates a very concrete way in which a simulation interacting with the real world fails to do its job not because it is missing “causal powers of the brain” (cf. Searle 1981 [1980]) or other matters debated from the proverbial armchair. I have in mind debates about AI in which one party claims

the interaction with the world is important saying not in the way you'd think. In principle, investigation in this area might give more substance to debates over "the robot reply".

But it gets worse. In the above I have discussed a metaphysical case, one in which the very properties of simulation have the real world "leak in" in a very concrete fashion and so alter irrevocably the simulation's characteristics. I turn now to the epistemological lessons raised by the above metaphysical consideration.

Section 3: Epistemological Lessons

I anticipate that a critic will just say that this is banal - we already know that simulation is not perfect, and so forth. We also, she would say, know that millisecond level timing is not possible inside a VM because we have (presumably) read VMWare's paper on the subject (VMWare 2005). So we shouldn't be surprised that ping times are a problem from within a virtual machine. So we shouldn't be surprised that using a virtualized witness server will also create problems. Maybe so, but trying to find the problem when it occurs (after all, witnessing won't fail if pings are off a few milliseconds) raises an interesting epistemological problem. This issue concerns how we come to know what is going "inside" a computer system. If your laptop is emitting black smoke from its optical drive, chances are it is broken. But suppose something else might be more subtly wrong? How does one find out about it? Often times these days the way systems administrators find out about problems is to rely on log files produced by the operating system and applications. Here the various components of the computer system record something that "happened to them", so an extremely banal but important point for what follows is: errors written to an error log have to be provoked by a

condition that the program writing the log has been designed to detect and report. (This may be the perennial error of the sort “an unexpected error occurred”, but this catch-all condition still has to be sensed and recorded.)

So in a very real sense much (at least initial) computer monitoring is precisely of the “black box” kind. And thus I have (at last) gotten to the title of this paper. **We see through an event log darkly**. This slogan might suggest the problem is the computing analogue to the undermining of theory by evidence. It is not quite the usual Duhem-Quine problem, however, as it is not a matter of having insufficient evidence, but rather (at least at first glance) a version of the “experimenter’s regress” (Collins 1985) which can be viewed as a special or related case. The internal tools used to tell us about what is wrong with (say) our server set up presuppose that they are being used in a real environment. It is generally not possible for software to discover that it is running in a VM³ and so all the error messages are what one would get in perhaps different circumstances in a real environment. All our tools are in danger of misreporting in various odd fashions. For example, ping inside a VM generally reports the correct value. In the case of the network situation I described with the database mirroring, 1 ms is the approximate usual latency. VMWare does try to “play tricks” so that this timing works out

³ There are some minor exceptions to this which are interesting but do not affect the **purpose** of virtualization/emulation, which is precisely to replicate the virtualized/emulated computing system. This goal, however useful, is now being reconsidered for VMWare and Microsoft’s Virtual PC solution because of the problems of the sort I have been discussing. This creates other interesting problems for another discussion.

appropriately (VMWare 2005). However, this is not perfect, and so once in a while the latency is absurd - sometimes well into the hundreds or thousands of milliseconds.

To make matters worse, a sort of “reverse” analogue of the subjectivist misinterpretation (Bunge 1967) of the uncertainty principle of quantum mechanics applies here. Unlike in the quantum case, where the spread of values is purportedly caused by the interaction with “the observer interfering with a system” but rather instead is inherent to the system itself, the latency considerations of a VM are actually made closer to optimum and “expected behaviour” by observing the system. The mechanism seems⁴ to be this: a relatively idle VM is given less time slices by its host. Then along comes some use which requires timing - perhaps message passing, perhaps database witnessing - and the timing is worse because the VM receives less timeslices. Then the host “notices” that there is a task running (e.g. the witness activity) and so gives more time. So the next run might (depending on the host scheduling) perform better. It might perform better still if the administrators notice that something has gone wrong by noticing huge timeout or other latency problems. But investigating the problem (e.g. by remote desktop-ing to the server), pinging it, etc. ramps up the time slices allocated to the VM and thus the latency and so the other problems become less and less frequent. This sequence of events is such that by the time one gets around to calling the network support people to test the network things are back to more or less normal.

⁴ This explanation for the behaviour of the virtual machines is a conjecture of mine supported also by investigations of my colleague, Michael Goit. I am indebted to him for discussions on this issue.

So, we have two epistemological puzzles: a regress problem and an “interaction between observer and observed”. The two have various solutions within the literature (Franklin 1998, 2007; Bunge 1967) in the case of the philosophy of science. But what are the solutions here? In the case of the second, there is no objective version of a theory which shows the subjectivist misinterpretation to be wrong. But there is something of an analogous solution to the “event log regress” as there is to the experimenter’s regress. I end by briefly sketching this.

Franklin’s approach functions by showing that we were too hasty in assuming that all the epistemic resources are “on the inside”. Some are, in fact, on the outside. One of these is what I will call “consilience” after Wilson (1998) or what Bunge (2003) calls “convergence”. This is what occurs when independent lines of evidence mutually support each other. To use Franklin’s examples, an instrument has to be capable of independent verification of what we already know - e.g. a machine looking for the charge of quarks better not report a continuous charge distribution as we know from Milikan and so on that this is wrong. Consilience then can provide additional support (never conclusively - but final certainty is a lost hope anywhere). VMWare behaving poorly in the cases I’ve discussed can be checked not only by pings but by measuring time an application takes to respond. For example, an application designed to work in the environment I’ve discussed may visibly take too long to do something, might time out, etc. So in this case a human (in the first case) forms part of the integrated computing system and checks; in the second an independently written application with our own knowledge of how it reports errors, etc.

In addition to consilience, we also have different ways of measuring the same value - for example we can use not only Windows ping but also the ping available to Orion, a network monitoring tool. This is somewhat banal: it is sort of analogous to using several different sorts of rulers and measuring tape. It is not quite that simple, as monitoring tools (at least in many environments) are also in different subnets than the workstations I would ordinarily ping from.

One can also measure latency (not network, this time, but over all) by pinging the loopback address of the server in question. For example, while remote-desktop connected to server X, simply ping 127.0.0.1. If the hypothesis that “time slice stealing” is correct, this will, from time to time, also report values which do not correspond to the load of the machine. I have confirmed this hypothesis. We should also be Popperian to some degree and attempt to falsify it by making sure that the timing problems do not occur in non-VM contexts. This can be done by (for example) moving the witness server I mentioned to a real machine temporarily. One can even simulate load by putting the witness server on a machine used for other purposes and, indeed one which is more burdened. As it happens, the hypothesis that load is the trouble is refuted for the time being, in our case. But these are only sketches of the beginning of an investigation. I thus conclude and leave details for another time.

Section 4: Conclusion

I discussed very brief several solutions to the “log regress” and did not propose any to the problem of interaction I discussed. Nor do I have any easy fixes for the metaphysical

problem I also raised. However, my purpose was merely to indicate a fruitful line of research and hopefully these inquiries can further pursued.

References

Bunge, Mario. (ed.) 1967. *Quantum Theory and Reality*. New York: Springer-Verlag.

Bunge, Mario. 2003. *Emergence and Convergence: Qualitative Novelty and the Unity of Knowledge*. Toronto: University of Toronto Press.

Collins, Harry. 1985. *Changing Order*. London: Sage.

Franklin, Allan. 1998. "Avoiding the Experimenter's Regress". In Koertge, Noretta (ed.) *A House Built on Sand: Exposing Postmodernist Myths About Science*. Oxford: Oxford University Press.

Franklin, Allan. 2007. "Experiment in Physics". In *The Stanford Encyclopedia of Philosophy (Fall 2007 Edition)*, Edward N. Zalta (ed.), URL = <http://plato.stanford.edu/archives/fall2007/entries/physics-experiment/>.

Hofstadter, Douglas and Dennett, Daniel. (eds.) 1981. *The Mind's I: Fantasies and Reflections on Self and Soul*. New York: Bantam Books.

Lehtinen, Aki and Kuorikoski, Jaakko. 2007. "Computing the Perfect Model: Why Do Economists Shun Simulation". *Philosophy of Science*, vol. 74. no. 3.

Searle, John. 1981 (1980). "Minds, Brains and Programs". Reprinted in Hofstadter and Dennett 1980.

Turing, Alan. 1936. "On Computable Numbers, With an Application to the Entscheidungsproblem." Reprinted in Davis, Martin (ed.) 1964. *The Undecidable*. Hewlett: Raven Press.

VMWare. 2005. "Time Keeping in VMWare Virtual Machines". URL = http://www.vmware.com/pdf/vmware_timekeeping.pdf. Retrieved January 20 2008.

Wilson, Edward. 1998. *Consilience: The Unity of Knowledge*. New York: Knopf.