One-Click Time Travel

Mahadev Satyanarayanan, Gloriana St. Clair[‡],
Benjamin Gilbert, Yoshihisa Abe, Jan Harkes, Dan Ryan[‡],
Erika Linke[‡], Keith Webster[‡]

June 2015
CMU-CS-15-115

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[‡]CMU Libraries

Abstract

We describe a system called *Olive* that freezes and precisely reproduces the environment necessary to execute software long after its creation. It uses virtual machine (VM) technology to encapsulate legacy software, complete with all its software dependencies. This legacy world can be completely closed-source: there is no requirement for availability of source code, nor a requirement for recompilation or relinking. The user experience is similar to viewing a YouTube video. A click on a web page launches the VM at an edge node on the Internet. As the VM executes, its contents are demand-paged from an unmodified web server. History-based prefetching is able to sustain acceptable user experience even over last-mile networks.

This work was supported by the Alfred P. Sloan Foundation and the Institute of Museum and Library Sciences. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and should not be attributed to Carnegie Mellon University or the funding sources.

Keywords: scientific reproducibility, virtual machines, demand paging, caching, prefetching, software archiving, software obsolescence, software maintenance, software forensics

1 Software in Science

Preserving software in ready-to-run form over long periods of time is difficult. Today, you can view the hardware of many old computers in museums, but you cannot easily run their software. Unfortunately, the need for long-term software preservation is real and growing. This is easiest to see in the context of scientific reproducibility.

At the heart of the scientific method is the ability to reproduce previously-reported results. Lowering barriers to the execution of archived software is important because high barriers discourage independent validation. The lowest conceivable barrier is one-click execution, much like viewing a `.pdf` document on a web page today.

The role of software in the scientific method is illustrated by a recent controversy [30]. In early 2010, Reinhart and Rogoff published an analysis of economic data spanning many countries [34, 35]. Herndon et al [20] refuted their findings in 2013 by discovering an error in their calculations. The significance of the error was described as follows [33]:

> "The Reinhart-Rogoff research is best known for its result that, across a broad range of countries and historical periods, economic growth declines dramatically when a country's level of public debt exceeds 90 per cent of gross domestic product.
> ...
> When we performed accurate recalculations using their dataset, we found that, when countries' debt-to-GDP ratio exceeds 90 per cent, average growth is 2.2 per cent, not -0.1 per cent."

The controversy continues, but regardless of how it is eventually resolved, there is no denying the central role of software (in this case, a Microsoft Excel spreadsheet) in the original analysis, its refutation and its eventual resolution.

Today, the use of software is pervasive in virtually all areas of scholarship. This includes physics, chemistry, biology, engineering, economics, political science and the humanities. Examples of software used for scholarly purposes include data analysis tools to slice and dice raw data, zoomable visualization tools that enable results to be viewed at many levels of abstraction, and simulation models written in a variety of programming languages and using a wide range of supporting libraries and reference data sets. Such software is central, not peripheral, to the discovery and reporting of new results today. Raw scientific data is often of limited value unless it is accompanied by the uniquely customized software to decode, interpret, analyze and display that data.

In the Reinhart-Rogoff example, there was no difficulty in obtaining the software necessary to perform the recalculations. Only three years had elapsed since the original publication of results, and the same version of Microsoft Excel continued to be in widespread use. Imagine, however, that the recalculations were attempted by a researcher 30 years later. Would Microsoft Excel still be in use? Would it accept the data format used by the original researchers? Would its calculations be identical in every respect (including, for example, handling of rounding errors) to those obtained by the original researchers? What if Microsoft goes out of business ten years after the original publication of results, and the Windows environment (which is needed to run Excel) ceases to be in use? As these questions suggest, our growing dependence on software in scientific research introduces new challenges to the premise of reproducibility that is the bedrock of science. Unless these challenges are addressed, our ability to re-validate published results will evaporate over time.

2 Execution Fidelity

Precise reproduction of software execution, which we call *execution fidelity,* is a complex problem in which many moving parts must all be perfectly aligned for a solution. Preserving this alignment over space and time is difficult. Many things can change: the hardware, the operating system, dynamically linked libraries, configuration and user preference specifications, geographic location, execution timing, and so on. Even a single change may hurt fidelity or completely break execution.

Unfortunately, the available mechanisms for enforcing execution fidelity are weak. Most software distribution today takes the form of *install packages,* typically in binary form but sometimes in source form. The act of installing a package involves checking for a wide range of dependencies, discovering missing components, and ensuring that the transitive closure of dependencies involving these components is addressed. Tools have been developed to simplify and partially automate these steps. However, the process still involves considerable skill and knowledge, remains failure-prone, and typically involves substantial time and effort.

These difficulties loom large to any researcher who attempts to re-validate old scientific results. Software install packages themselves are static content, and can be archived in a digital library using the same mechanisms that are used to archive scientific data. However, the chances of successfully installing and executing this software in the distant future are low. In addition to all of the software installation challenges mentioned above, there is the additional difficulty that the passage of time makes hardware and software environments obsolete. The chances of finding compatible hardware and operating system on which to even attempt an install become vanishingly small over time scales of decades. These challenges have long stymied efforts to archive executable content [7, 8, 28].

3 Olive Design and Implementation

We have created a system called *Olive* that leverages VM technology to encapsulate and deliver a bit-exact, pre-packaged execution environment over the Internet. This environment can be completely closed-source: there is no requirement for availability of source code, nor a requirement for recompilation or relinking. The user experience is similar to viewing a YouTube video. A click on a web page launches the VM at an edge node on the Internet. This could be the user's desktop or laptop, or it could be a private cloud or cloudlet close to the user and connected via a remote desktop protocol. As the VM executes, its contents are demand-paged from an unmodified web server. History-based prefetching is able to sustain acceptable user experience even over last-mile networks.

Figure 1 illustrates the conceptual structure of an Olive client. At the bottom (1 and 2) is standard Intel x86 desktop or laptop hardware running Linux (generically called the "host operating system"). Layered above this (3) is an Olive component called *VMNetX* that implements caching and prefetching of VM images over the Internet. VMNetX presents the illusion of a fully assembled VM image to the VMM layer above, which virtualizes the x86 host hardware. We use KVM/QEMU as our VMM. Layers 5 through 8 in Figure 1 are encapsulated within the archival VM image that is streamed from Olive servers. The lowest of these layers (5) is a hardware emulator that presents the illusion of now-obsolete hardware

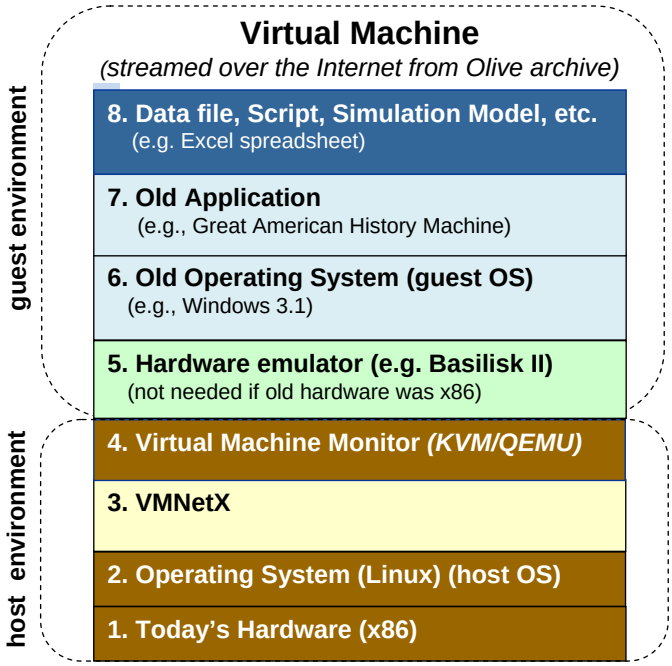


Figure 1: Abstract Olive Client Structure

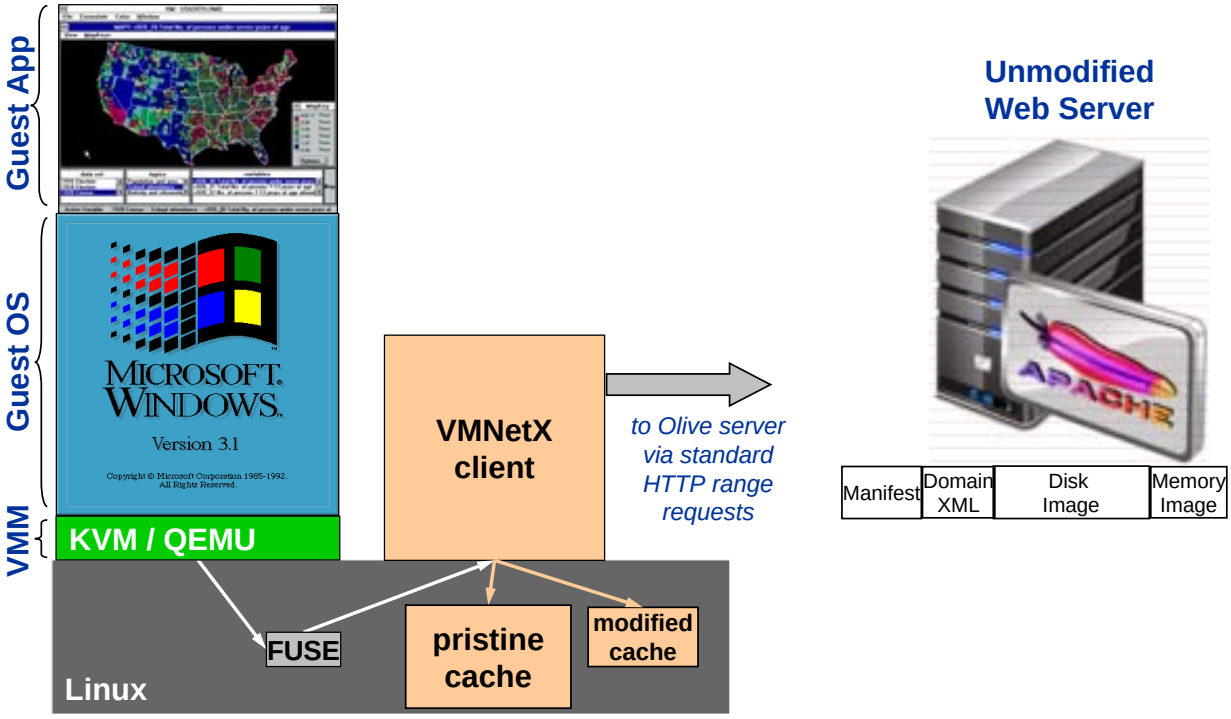


Figure 2: Olive Architecture

(such as Motorola 68040). This layer can be omitted if the archived environment targets x86 hardware. Layer 6 is the archived operating system (generically called the "guest" operating system). The virtual disk of the VM is managed by the guest operating system, and appears as a local file system to higher layers.

Layer 7, which represents the archived application (such as the Great American History Machine) is the focal point of interest in archiving. It is to support execution of this application with high fidelity that the entire edifice shown in Figure 1 is necessary. Layer 8 represents input that is provided to the archived application. In the Reinhart-Rogoff example, Layer 8 would be the original Excel spreadsheet that was used in their analysis. Layer 7 would be the version of Excel that they used. In a different situation, such as examining an old archived engineering drawing, Layer 7 might be the AutoCAD application and Layer 8 would be the input files to AutoCAD that represent the drawing. Alternatively, Layer 8 may be placed on an external data source such as a distributed file system and exposed to the guest OS as a virtual storage device or a file share.

Figure 2 shows how the abstract layers shown in Figure 1 are mapped to the Olive architecture. Layers 8 through 5 are encapsulated within the VM instance shown on the left. Layer 4 (KVM/QEMU) is explicitly shown in Figure 2. Layer 3 (VMNetX) maps to the user-level process and file caches ("pristine" and "modified"). As the VM instance executes, it may access parts of its VM image that have not been cached yet. VMNetX services these cache misses using HTTP range requests to a standard Web server such as Apache. The "web page" in this case is a large file on the server that contains all components of the VM image, including its disk image, its memory image, and its hardware configuration. The partitioning of pristine and modified VM state into separate caches makes it easier to ensure that a fresh launch of a VM instance always starts with the bit-exact VM image in the cloud. All cached VM state in the pristine cache corresponds to the VM image in the cloud. As a VM instance executes, some of this state may be modified. If that modified state is written out to disk, it goes into the modified cache.

To support non-Linux clients (such as Windows desktops and laptops, as well as Android tablets), the entire client structure shown in Figure 2 can be executed on a nearby cloudlet or private cloud. We use the SPICE remote desktop protocol [38] for thin client user interactions with the VM instance, as shown in Figure 3. Of course, network connectivity has to be of sufficiently low latency and adequate bandwidth for a remote desktop protocol to provide a good user experience. As long as these criteria are met, our implementation offers considerable runtime flexibility in the physical locations of user and VM execution sites.

Last-mile networks such as 4G cellular networks pose special challenges for Olive. Their low bandwidth and high latency make demand paging of Olive VMs over the Internet unacceptably slow. We have conducted experiments with history-based prefetching of VM state over last-mile networks in an experimental version of Olive called *vTube* [1]. To generate accurate prefetching hints, vTube uses fine-grained analysis of disk and memory state access traces from previous executions. Our preliminary results show that despite wide variances from execution to execution and from user to user, vTube can identify short segments of state access that once activated, are exceptionally stable across multiple executions and can thus provide high-quality predictive hints. Figure 4 from Abe et al [1] shows sample runs of vTube for a VM encapsulating the game Riven over various last-mile networks. For each run, black continuous lines indicate periods of uninterrupted execution, while gray interruptions

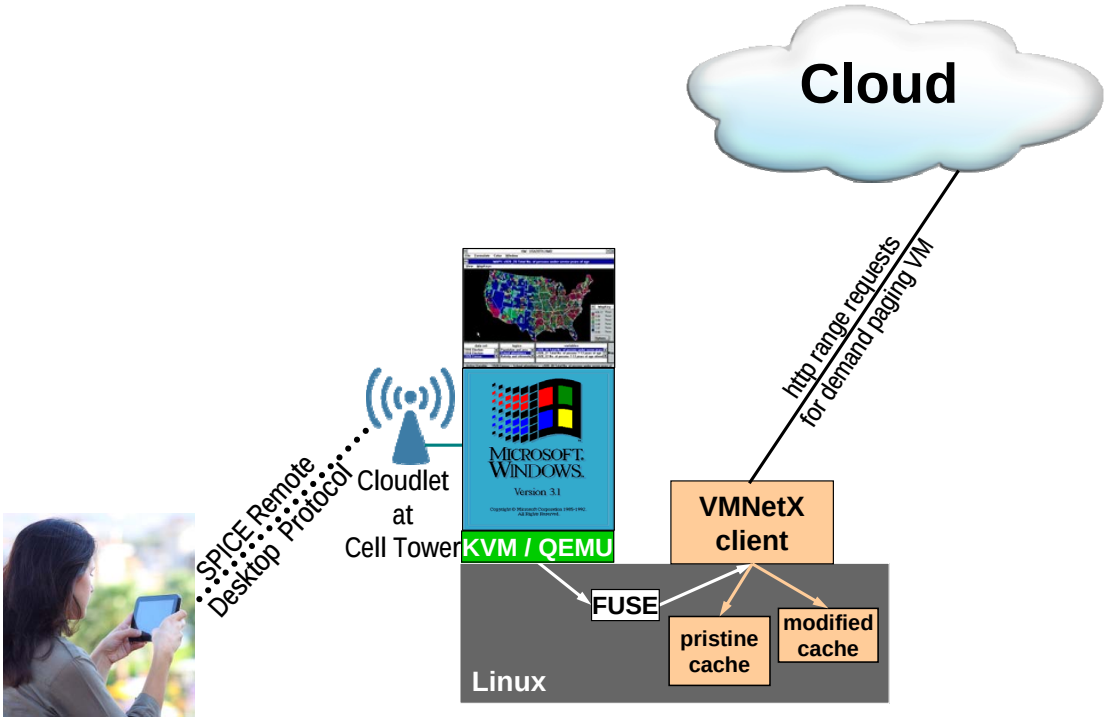


Figure 3: Olive Thin Client Access

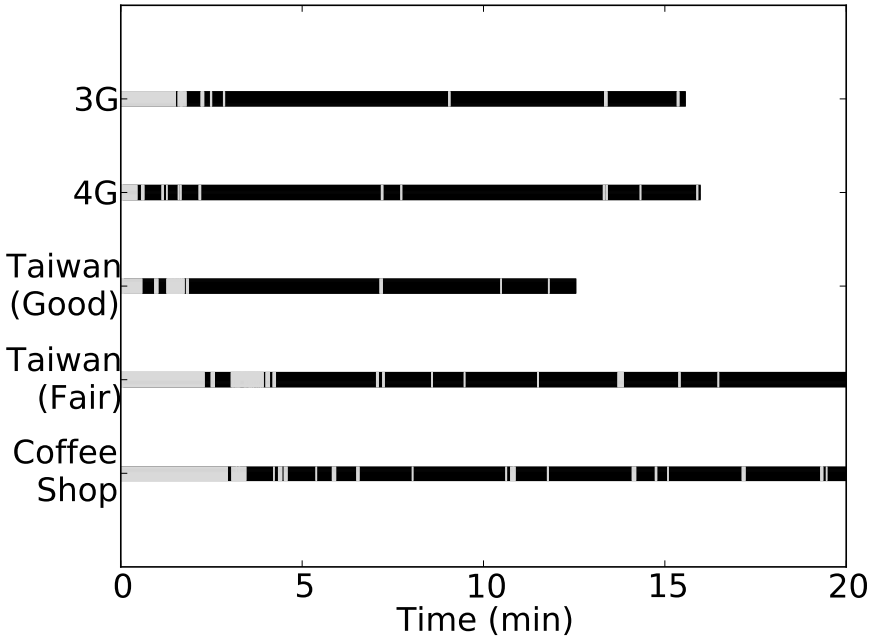


Figure 4: VM Prefetching in vTube (Source: [1])

indicate when VM execution is stalled either due to explicit buffering periods or memory demand fetches. Qualitatively, the user experience in vTube during these experiments is comparable to viewing video over a last-mile network.

4 Why Hardware Virtualization?

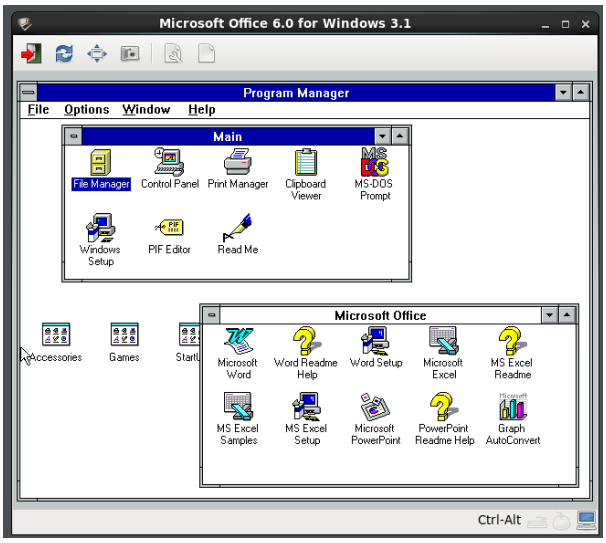
In the context of Olive, virtualization refers specifically to *hardware virtualization* of the Intel x86 architecture, and the term "VM" refers to a virtualized x86 machine. Today, this hardware architecture is dominant and is efficiently virtualized using Intel's VT extensions. Olive benefits indirectly from the many efforts in academia and industry that are aimed at improving the performance and functionality of VM-based systems for cloud computing. As described in Section 5, Olive VMs can archive software written for other hardware architectures. That involves an additional layer of emulation that is nested within the x86 VM, and thus incurs additional runtime overhead. Specific benefits arise from our choice of x86 as the virtualization target rather than software virtualization alternatives such as the Java Virtual Machine (JVM) [23] or the Dalvik Virtual Machine [10].

First, the VM interface is compatible with legacy operating systems and their valuable ecosystems of applications. The ability to sustain these ecosystems without code modifications is a powerful advantage of VMs. The ecosystems supported by software virtualization tend to be much smaller. For example, a JVM is only valuable in supporting applications that compile to Java bytecode. In contrast, a VM is language-agnostic and OS-agnostic. In fact, a JVM can be part of VM's ecosystem. Hardware virtualization can thus subsume software virtualization.

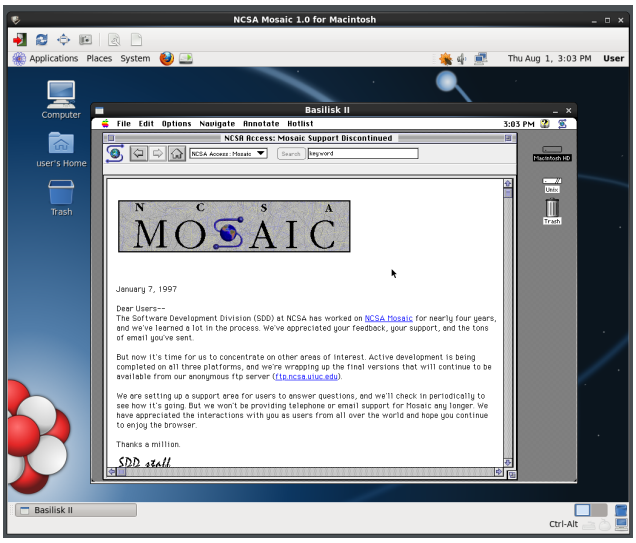
Second, a VM interface is *narrow* and *stable* relative to typical software interfaces. These attributes help to preserve execution fidelity over long periods of time. The stability of a VM interface arises from the fact that the hardware it emulates itself evolves very slowly and almost always in an upward-compatible manner. In contrast, the pliability of software results in more rapid evolution and obsolescence of interfaces. Keeping up with these changes requires high software maintenance effort. Pliability also leads to widening of narrow interfaces over time. Over time, the burden of sustaining a wide interface compromises execution fidelity.

This line of reasoning leads to an approach in which VMs play a role for archiving executable content that is analogous to the role played by a standardized document reader format such as `pdf` today. There may be many alternative paths to producing a VM, but once produced that VM can be saved in an Internet library and viewed on demand by anyone with appropriate access privileges.

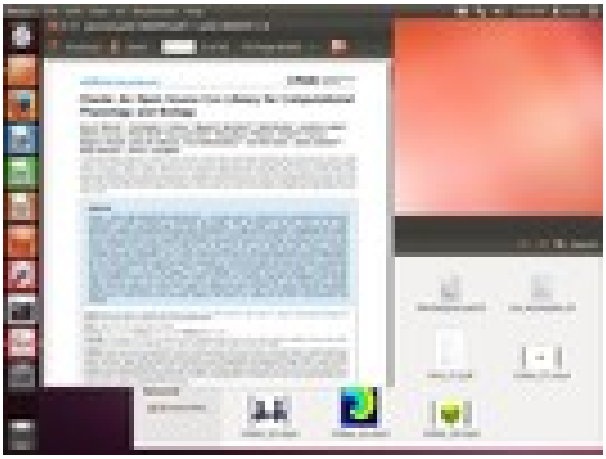
Lightweight virtualization approaches such as Linux containers [6] and Docker [15] have gained popularity recently, and offer possible alternatives to the use of VMs. Unfortunately, they were not designed for long-term archiving and would require active maintenance of considerably more software infrastructure than a VM-based Olive. Our desire to preserve complete environments (both Linux and non-Linux, including Windows and MacOS) for decades-long periods favors the use of VMs.



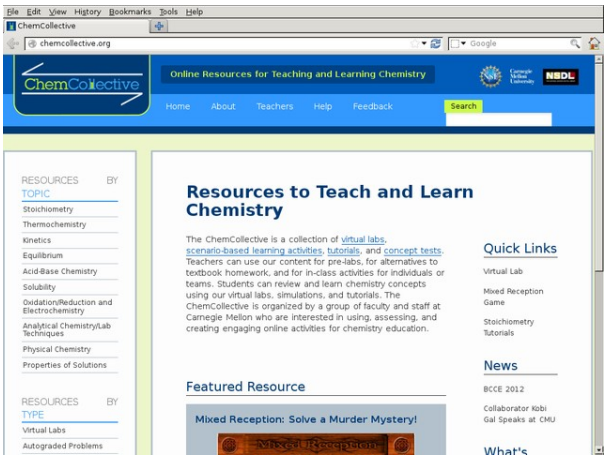
(a) Microsoft Office 6.0 for Windows 3.1 on x86



(b) Mosaic Browser for MacOS 7.5 on Motorola 68040



(c) Chaste 3.1 for Scientific Linux on x86



(d) ChemCollective for Scientific Linux on x86

Figure 5: Example Screenshots of Olive VMs

5 Olive Status (http://olivearchive.org)

Olive contains over 15 VMs today, and the collection continues to grow. It includes operating systems and applications from the early 1980s through 2013. Because of software licensing restrictions, the collection is currently accessible over the Internet only to a limited number of research collaborators. We are in active dialog with software vendors to obtain the necessary permissions to allow a broader user community to access the collection. A summary of the full collection can be found at `https://olivearchive.org/docs/collection/`. To give a taste of the collection, and to highlight the diverse content that can be archived in Olive, we briefly describe four of these VMs below.

Microsoft Office 6.0: Figure 5(a) shows a screenshot of this VM, containing Word, Excel and PowerPoint for Windows 3.1. If Reinhart and Rogoff had published their controversial paper [34] in the 1993-94 timeframe, this is the VM that you would need to re-validate their results today.

NCSA Mosaic: As the world's first widely-used web browser dating back to 1992-93, Mosaic has a unique historical status. This VM, whose screenshot is shown in Figure 5(b), is also interesting for a second reason. The version of Mosaic that it encapsulates was written for the Apple MacOS 7.5 operating system on Motorola 68040 hardware. The VM also encapsulates Basilisk II, an open source hardware emulator for Motorola 68040 on modern Intel x86 hardware running Linux. The bootable disk image of MacOS 7.5 with Mosaic is stored as a file in the virtual file system of the outer Linux guest. In spite of two levels of virtualization, performance is acceptable because modern hardware is so much faster than the original Apple hardware. Pointing the Mosaic browser at modern web sites is instructive. Since Mosaic predates web technologies such as JavaScript, HTTP 1.1, Cascading Style Sheets, and HTML5 it is unable to render content from modern web sites. It is, however, capable of rendering web pages from some older Internet sites.

Chaste 3.1: This computational biology VM (Figure 5(c)) illustrates the value of Olive in stably preserving the environment needed to build from source code. Chaste (Cancer, Heart and Soft Tissue Environment) is a simulation package for computationally demanding problems in biology and physiology that was developed at Oxford Unviersity. This particular release of Chaste was packaged with a paper that was published in March 2013 [29]. Even though it is less than two years since the paper was published, the source code no longer compiles on current Linux releases. A number of source code changes are needed before compliation succeeds. This problem will grow worse over time, and represents exactly the kind of barrier to entry for scientific reproducibility that was mentioned earlier. The Chaste VM contains a frozen Linux environment in which the Chaste code successfully compiles. It also contains example data that was published with the paper. Running Chaste on this data produces videos showing visualizations of certain muscle functions. With high confidence of success, a future researcher who wishes to explore a modification to the published software can edit the code in the VM, then compile and run it, and finally view the generated videos.

ChemCollective: This VM (Figure 5(d)) illustrates how Olive can be used to archive frozen snapshots of a cloud service in ready-to-execute form. The ChemCollective is a web-based service for teaching and self-learning chemistry. It contains a collection of virtual labs, scenario-based learning activities, tutorials, and concept tests. Teachers can use the content for pre-labs, for alternatives to textbook homework, and for in-class activities for

individuals or teams. The live web site is constantly evolving, as new material is added and old material is updated. This VM represents a frozen snapshot of the web service at one point in time, and contains the complete static data of the web site, an application server, a web server, and a browser. The VM's hostname to IP mapping has been modified to redirect all ChemCollective references back to the local host. This ensures that ChemCollective links traversed by the browser within the VM will map to the frozen ChemCollective service within the VM rather than to the live ChemCollective web site.

6 Related Work

Olive complements, but does not duplicate, many other efforts that are aimed at improving scientific reproducibility. Closest in spirit are RunMyCode.org [39] and ReproZip [5].

RunMyCode.org is a cloud-based service that enables authors to create companion web pages for published scientific papers. The service accepts code written in C++, FORTRAN, MATLAB, R and RATS. An IT staff team associated with the cloud service performs safety and executability checks on submitted code before it is accepted. Once a companion web page is created, users can submit scripts that use code from that page. These scripts are executed on a cluster in the cloud, and the results are returned to the user. Compared to RunMyCode.org, Olive aims at a much lower level of abstraction and focuses on preserving executability over a timescale of decades.

ReproZip [5] aims to simplify the re-creation of computations described in research publications. This tool automatically captures the provenance of an experiment and creates a package of all its library dependencies as well as its workflow specification. The package can then be disseminated or archived. ReproZip is specifically designed for Linux environments. The VM encapsulation used by Olive may offer a way to extend ReproZip to non-Linux environments.

There has been a considerable amount of effort by the scientific community in creating workflow management software. Examples include the ISI Pegasus framework [13], the IPython interactive shell [32], the Taverna Workbench [21], the Sumatra management tool [11, 12], the Galaxy tool suite [4, 18], the Madagascar platform [17], VisTrails [3], and verifiable visualizations [16]. There has also been significant effort in creating data sharing tools and repositories such as Dexy [14], Duraspace [27], and DataVerse [9]. Although these efforts do not overlap with Olive, they may be able to leverage its functionality. For example, a workflow tool could be extended to produce a snapshot of its state as an Olive VM. This could be useful for dissemination, and to serve as a permanent easy-to-run marker in that workflow.

As mentioned earlier, it may be helpful to think of an Olive VM as similar to a `.pdf` file in document production. One uses tools such as Latex or Microsoft Word for authoring. The evolution of the document can be captured using CVS, git, or Word's internal change tracking mechanism. However, a landmark version of the document can also be saved in a `.pdf` file for convenient one-click viewing.

The term *emulation* is used by the digital library community to describe the Olive approach to archiving. The essence of this approach is precise re-creation of the external dependencies of a piece of software, combined with precise behavioral replication of the un-

derlying hardware. The promise of emulation as an archiving strategy was first articulated by Rothenberg [36]. A Universal Virtual Computer that could serve as a fixed execution engine for archiving was described by Lorie [24]. An x86 emulator called *Dioscuri* and its use in emulating MS-DOS programs was demonstrated by van der Hoeven et al [40]. Olive makes several contributions relative to these prior efforts from the digital library community. First, it uses virtualization technology from the world of cloud computing, thus leveraging the enormous investments being made by that community towards improving performance, fidelity and scalability. Second, Olive addresses the problem of efficient execution of large archived VMs over the Internet. By allowing clean separation of VM storage site from VM execution site, it offers control and flexibility for real-world deployments. Third, Olive has gone well beyond conceptualization to provide validation on over 15 VMs ranging from MS-DOS environments of the mid-1980s to present-day Linux and Windows software.

7 Challenges Ahead

Olive is at an exciting point in its early evolution, but it is far from being a fully-deployed system. In addition to the software licensing issues mentioned earlier, there are numerous technical challenges to be addressed. We mention just three of these below.

Access to External Data Sets: Since Olive does not yet support access to external data sources from applications within a VM, data has to be manually copied in. This is inconvenient and error-prone. It also limits data to the size of VM's virtual disk. Olive will need to simplify and streamline access to large data sets that are often required in scientific computing. We envision these data sets being placed in a external distributed file system such as AFS [31], Lustre [25], or MagFS [26].

Parallelism and Compute Clusters: The current Olive prototype can exploit multi-core parallelism, but it is not possible to change the number of cores available to the guest. We expect this to become a common requirement in the future, as VMs that were archived a long time ago are launched on modern many-core machines. Also relating to parallelism is the need to exploit cluster-level parallelism for large scientific applications. Today, this involves extensive manual configuration of multiple VMs using VLANs. This is an error-prone and slow workflow with poor reproducibility. One-click launch of an entire ensemble of VMs, correctly interconnected, would be a great simplification.

GPU Acceleration: Beyond the original motivation for graphics, the SIMD parallelism of GPUs has been leveraged by the scientific community for many computations in simulation and finite element modeling. Virtualizing GPUs has proven difficult because there is no standardized external interface for them. There have been many efforts at GPU virtualization [2, 19, 22, 37], but none has yet emerged dominant. Although GPU support in Olive is likely to be messy, it is an effort that is too important to shirk.

8 Conclusion

Executable content ranging from simulation models to visualization tools plays an increasingly important role in scholarly research. The ability to archive these artifacts for posterity

would be an important transformative step. Imagine being able to reach back across time to execute the simulation model of a long-dead scientist on new data that you have just acquired. What do the results suggest? Would they have changed the conclusions of that scientist? Although you aren't quite bringing the scientist back to life, you are collaborating with that person in a way that was not possible until the emergence of Olive. We look forward to advancing scholarship in many fields through Olive.

References

[1] Y. Abe, R. Geambasu, K. Joshi, A. Lagar-Cavilla, and M. Satyanarayanan. vTube: Efficient Streaming of Virtual Appliances Over Last-Mile Networks. In *Proceedings of the ACM Symposium on Cloud Computing*, Santa Clara, CA, October 2013.

[2] A. Amiri Sani, K. Boos, S. Qin, and L. Zhong. I/O Paravirtualization at the Device File Boundary. In *Proceedings of ACM ASPLOS*, March 2014.

[3] L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. E. Scheidegger, C. T. Silva, , and H. T. Vo. VisTrails: Enabling Interactive Multiple-View Visualizations. In *Proceedings of IEEE Visualization*, pages 135–142, 2005.

[4] D. Blankenberg, G. von Kuster, E. Bouvier, B. Baker, E. Afgan, N. Stoler, B. Rebolledo-Jaramillo, The Galaxy Team, J. Taylor, and A. Nekrutenko. Dissemination of scientific software with Galaxy ToolShed. *Genome Biology*, 15:403, February 2014.

[5] F. Chirigati, D. Shasha, and J. Freire. ReproZip: Using Provenance to Support Computational Reproducibility. In *Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance*, Lombard, Illinois, 2013.

[6] Linux Containers. `https://linuxcontainers.org/`. Accessed on January 9, 2015.

[7] P. Conway. Preservation in the Digital World. `http://www.clir.org/pubs/reports/conway2/`, March 1996.

[8] P. Conway. Preservation in the Age of Google: Digitization, Digital Preservation, and Dilemmas. *Library Quarterly*, 80(1), 2010.

[9] M. Crosas. The Dataverse Network: An Open-Source Application for Sharing, Discovering and Preserving Data. *D-Lib Magazine*, 17(1/2), January/February 2011.

[10] Dalvik (software). `http://en.wikipedia.org/wiki/Dalvik_%28software%29`. Accessed on May 2, 2014.

[11] A. Davison. Automated capture of experiment context for easier reproducibility in computational research. *Computing in Science and Engineering*, 14, 2012.

[12] A. Davison, M. Mattioni, D. Samarkanov, and B. Teleczuk. Sumatra: A Toolkit for Reproducible Research. In V. Stodden, F. Leisch, and R. Peng, editors, *Implementing Reproducible Research*, pages 57–79. Chapman and Hall/CRC, 2014.

[13] E. Deelman, G. Singh, M. hui Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13, 2005.

[14] dexy: The Most Powerful, Flexible Documentation Tool Ever. `http://dexy.it/`.

[15] Build, ship, and run any app, anywhere. `https://www.docker.com/`. Accessed on January 9, 2015.

[16] T. Etiene, C. Scheidegger, L. Nonato, M. Kirby, and C. Silva. Verifiable Visualization for Isosurface Extraction. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1227–1234, 2009.

[17] S. Fomel, P. Sava, I. Vlad, Y. Liu, and V. Bashkardin. Madagascar: open-source software project for multidimensional data analysis and reproducible computational experiments. *Journal of Open Research Software*, 1(1), 2013.

[18] J. Goecks, A. Nekrutenko, J. Taylor, and The Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8), August 2010.

[19] J. G. Hansen. Blink: Advanced Display Multiplexing for Virtualized Applications. In *Proceedings of ACM Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 2007.

[20] T. Herndon, M. Ash, and R. Pollin. Does High Public Debt Stifle Economic Growth? A Critique of Reinhart and Rogoff. Working Paper 322, Political Economy Research Institute, University of Massachussets Amherst, April 2013. `http://www.peri.umass.edu/236/hash/31e2ff374b6377b2ddec04deaa6388b1/publication/566/`.

[21] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34:729–732, 2006.

[22] H. Lagar-Cavilla, N. Tolia, M. Satyanarayanan, and E. de Lara. VMM-Independent Graphics Acceleration. In *Proceedings of the 3rd international Conference on Virtual Execution Environments*, San Diego, CA, June 2007.

[23] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification (2nd Edition)*. Prentice Hall, 1999.

[24] R. Lorie. Long Term Preservation of Digital Information. In *Proceedings of the Joint Conference on Digital Libraries*, Roanoake, VA, June 2001.

[25] Lustre File System. `http://lustre.org`. Accessed June 19, 2014.

[26] Maginatics Cloud Storage Platform. `https://maginatics.com/product/maginatics-cloud-storage-platform`. Accessed June 19, 2014.

[27] DuraSpace Offers DuraCloud Access To Internet2 Members, April 2012. `http://duraspace.org/node/1268`.

[28] B. Matthews, A. Shaon, J. Bicarreguil, and C. Jones. A Framework for Software Preservation. *The International Journal of Digital Curation*, 5(1), June 2010.

[29] G. R. Mirams, C. J. Arthurs, M. O. Bernabeu, R. Bordas, J. Cooper, A. Corrias, Y. Davit, S.-J. Dunn, A. G. Fletcher, D. G. Harvey, M. E. Marsh, J. M. Osborne, P. Pathmanathan, J. Pitt-Francis, J. Southern, N. Zemzemi, and D. J. Gavaghan. Chaste: An Open Source C++ Library for Computational Physiology and Biology. *PLoS Computational Biology*, 9(3), March 2013.

[30] P. Monaghan. 'They Said at First That They Hadn't Made a Spreadsheet Error, When They Had'. *The Chronicle of Higher Education*, April 2013. `https://chronicle.com/article/UMass-Graduate-Student-Talks/138763/`.

[31] OpenAFS. `http://openafs.org`. Accessed June 19, 2014.

[32] F. Perez and B. E. Granger. IPython: a System for Interactive Scientific Computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007.

[33] R. Pollin and M. Ash. Austerity after Reinhart and Rogoff. *Financial Times*, April 2013. `http://www.ft.com/cms/s/0/9e5107f8-a75c-11e2-9fbe-00144feabdc0.html#axzz2zLV7HuwS`.

[34] C. M. Reinhart and K. S. Rogoff. Growth in a Time of Debt. *American Economic Review*, 100(2):573–78, May 2010.

[35] C. M. Reinhart and K. S. Rogoff. Growth in a Time of Debt. Working Paper 15639, National Bureau of Economic Research, January 2010. `http://www.nber.org/papers/w15639`.

[36] J. Rothenberg. Ensuring the Longevity of Digital Documents. *Scientific American*, January 1995.

[37] L. Shi, H. Chen, and J. Sun. vCUDA: GPU Accelerated High Performance Computing in Virtual Machines. In *IEEE International Symposium on Parallel & Distributed Processing*, 2009.

[38] Spice. `http://www.spice-space.org/`.

[39] V. Stodden, C. Hurlin, and C. Perignon. RunMyCode.org: a novel dissemination and collaboration platform for executing published computational results. In *Analyzing and Improving Collaborative eScience with Social Networks (eSoN 12); Workshop with IEEE e-Science 2012*, Chicago, IL, USA, Oct. 2012. Also available at SSRN: `http://ssrn.com/abstract=2147710`.

[40] J. van der Hoeven, B. Lohman, and R. Verdegem. Emulation for Digital Preservation in Practice: The Results. *International Journal of Digital Curation*, 2(2), December 2007.