Collaborating with Executable Content Across Space and Time

Mahadev Satyanarayanan, Vasanth Bala[†], Gloriana St. Clair, Erika Linke

October 2011
CMU-CS-11-135

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

To appear in: Proceedings of the 7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom2011), October 2011, Orlando, FL
[†]IBM Research

Abstract

Executable content is of growing importance in many domains. How does one share and archive such content at Internet-scale for spatial and temporal collaboration? *Spatial collaboration* refers to the classic concept of user collaboration: two or more users who are at different Internet locations performing a task using shared context. *Temporal collaboration* refers to the archiving of context by one user and use of that context by another user, possibly many years or decades later. The term "shared context" has typically meant shared documents or a shared workspace such as a whiteboard. However, executable content forces us to think differently. Just specifying a standardized data format is not sufficient; one has to accurately reproduce *computation.* We observe that the precise encapsulation of computing state provided by a *virtual machine (VM)* may help us solve this problem. We can cope with large VM size through a streaming mechanism that demand fetches memory and disk state during execution. Based on our positive initial experience with VMs for archiving execution state, we propose the creation of *Olive,* an Internet ecosystem of curated VM image collections

This research was supported by the National Science Foundation (NSF) under grant numbers CNS-0833882 and CNS-0509004, an IBM Open Collaborative Research grant, and Intel. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily represent the views of the NSF, IBM, Intel, or Carnegie Mellon University.

Keywords: Olive, ISR, OpenISR, Internet Suspend/Resume, virtual machines, VM, VM streaming, cloud computing, digital library, digital archive, operating systems, file systems, software archaeology, software forensics

1 Introduction

Collaboration is defined as "the action of working with someone to produce or create something" [17]. *Shared context* is essential for successful collaboration. It typically takes the form of a shared workspace, document, machinery, or other tangible object. This leads to the kinds of scenarios we associate with collaboration: a group of users standing in front of a whiteboard; coauthors working on editing a document; mechanics troubleshooting an appliance; military planners studying a terrain model; and so on. The advent of the Internet has extended these scenarios to collaborators who are physically separated by considerable distances, leading to *collaboration across space.* Tools such as distributed workspaces, shared whiteboards, and collaborative authoring software, as well as control mechanisms for security and privacy have arisen from these roots.

Less explicitly recognized, but of equal impact and historical significance, is the concept of *collaboration across time.* A composer and a performer, though separated by centuries, are effectively collaborating on the creation of music. More generally, a long-dead author and a reader can collaborate on the mental transformation of the reader; these, in turn, can sometimes lead to more momentous real-world creations or transformations. Of course, there are fundamental limitations to collaboration across time. Unlike collaboration across space, where the collaborators can exert mutual influence on each other, a reader cannot reach back in time and get scientific authors' opinions on how newly available data would change their theories. [1]

The accurate preservation, easy discovery, and rapid retrieval of shared context to support collaboration over time has been the domain of the digital library community. Mechanisms such as databases, distributed storage systems, the Web, and search engines can be viewed as tools that support this effort. Underlying this effort is the fundamental assumption that shared context is in the form of *static content* such as a book or a musical score.

Today, an increasing fraction of the world's intellectual output is in the form of *executable content.* These include simulation models, tutoring systems, expert systems, data visualization tools, and so on. Even content that could be static (such as a company's product information Web page) is often dynamically generated through execution of code that customizes the content and appearance at runtime. What does it mean to preserve such content with perfect fidelity over time? There is no static object or collection of static objects that can serve as shared context to freeze and preserve. Rather, we need to *freeze and precisely reproduce the execution that dynamically produces the content.*

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.

The need for execution fidelity is not unique to the digital library community. In the domain of commercial software, it is a problem faced by every software vendor. There is a strong desire to package software in a manner that installs and works with minimal tinkering so that the *time to value* for a customer is as short as possible. This desire to avoid extensive installation effort argues for rigid constraints on the software. At the same time, vendors wish to target the largest possible market. This argues for rich flexibility and configurability of software at installation time, to address the many unique needs and constraints of different market niches and diverse individual customers. Reconciling these contradictory goals is a difficult challenge.

[1] As we shall see later in this paper, emerging technology may make it possible to partly alleviate this strong asymmetry.

Execution fidelity is also valuable when collaboration across space involves executable content. When help desk technicians assist remote customers, their ability to observe software behavior jointly in real time is likely to be more effective than having customers describe what they are seeing on their screens. Today, the most common class of tools for this purpose involves *remote desktop technology.* Examples include RDP in Windows [4] and VNC [19] in Linux, as well as Web-based commercial services such as WebEx [1] and GoToMyPC [2].

Unfortunately, the available mechanisms for ensuring 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 results in a long time-to-value metric for a customer.

These limitations directly affect collaboration across time. The install packages themselves are static content, and can be archived in a digital library with existing mechanisms. 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 by the digital library community to archive executable content [8, 9, 15].

For collaboration across space, today's use of remote desktop technology works reasonably well but faces fundamental challenges. For highly interactive applications such as games and data visualizations, the end-to-end network latency between collaborating parties limits the perception of perfect fidelity of execution. There is growing evidence that although Internet bandwidth will continue to improve over time, there are few economic incentives to keep end-to-end latency low. As discussed elsewhere [25, 26, 20], this limits the applicability of the remote desktop approach at Internet scale to a fairly narrow class of applications.

Is there a better way to ensure execution fidelity? Ideally, such a mechanism would simultaneously address the challenges of software distribution as well as collaboration with executable content over space and time. We present such a mechanism in the next section.

3 Virtual Machine Technology

A *virtual machine (VM)* is a computer architecture and instruction set emulator of such high accuracy that neither applications nor the operating system are able to detect the presence of the emulator. In other words, emulated execution is indistinguishable from execution on genuine hardware. Since the emulated hardware is usually of the same type as the bare hardware on which emulation is performed, we are effectively cloning a physical machine into multiple VMs.

Figure 1 illustrates the key abstractions in the realization of a VM. The *guest* layer represents an unmodified legacy computing environment, including one or more applications and the operating system. The *host* layer is responsible for virtualizing the bare hardware, including devices such as disk, network, display and so on. The host layer multiplexes the concurrent execution of multiple VMs, and ensures complete isolation of each VM's actions from other VMs. Each VM has execution context represented by a *VM monitor (VMM)* in Figure 1.

VMs were invented by IBM in the mid-1960s as a timesharing approach that enabled concurrent, cleanly-isolated system software development on a single mainframe by multiple programmers. Since main-

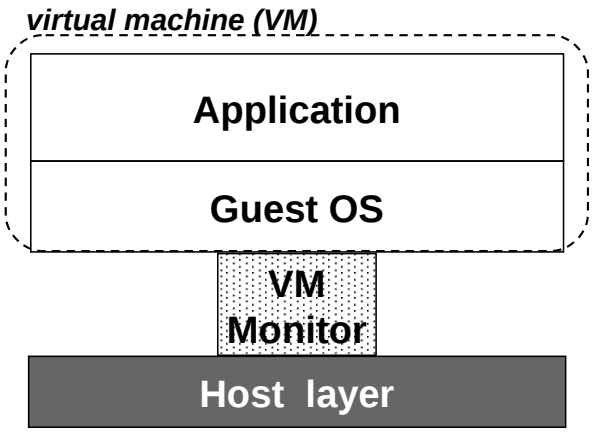


Figure 1: Basic VM Concepts

frame hardware of that era was expensive, VMs were a cost-effective approach to enhancing programmer productivity by providing a private "mainframe" for each developer rather than reserving dedicated time on real hardware. Accuracy of hardware emulation was paramount because the system software developed on a VM was intended for use in a mainframe operating system. That software had to work with negligible changes on the real hardware. The techniques for efficient and accurate hardware virtualization that were developed in this era have proved to be of lasting value. The accuracy of virtualization is a key reason for VMs being a potential solution to the problem of execution fidelity today.

By the early 1970s, VMs were being used for purposes well beyond those that motivated their invention. A commercial product, the VM/370 timesharing system [10], was highly successful and was in widespread use for well over a decade. Its descendants continue to be in use on IBM mainframes as z/VM today [5]. However, the emergence of the personal computer led to the eclipse of the VM as a computing abstraction by the late 1980s. Cheap PCs with a rigidly-controlled operating system (MS-DOS initially, and various versions of Windows later) displaced VMs as the most cost-effective way to deliver computing with high execution fidelity. Events have now come full circle. The emergence of cloud computing has restored lustre to the VM abstraction.[2] Large commercial investments are now being made in VM technology and VM-based systems.

Figure 2 illustrates the value proposition of the VM abstraction today. A large legacy world of software, including operating system software, represents a substantial intellectual and financial investment that has been already been made on existing computer hardware. This legacy world can be completely closed-source: there is no requirement for availability of source code, nor a requirement for recompilation or relinking. All that is needed is standard software installation, just as on real hardware. Beneath the implementation of the emulated hardware interface, there can be extensive innovation along many dimensions. This innovation is totally isolated from the idiosyncrasies of the legacy world above. In cloud computing, the freedom to innovate beneath the hardware interface allows the creation of valuable system management functionality such as server consolidation for energy savings and elastic computing for bursty server loads. More generally, it allows an enterprise to virtualize and outsource its entire data center to a remote third party such as Amazon Web Services [6].

Certain specific attributes of the VM abstraction accounts for its longevity and success. The hourglass shape in Figure 2 could depict the interface specification of any abstraction (thin waist), applications dependent on this interface (above), and implementations of this interface (below). More specifically, it could represent an execution engine such as the Java Virtual Machine (JVM) [14] or the Dalvik Virtual Machine

[2]The VMs in cloud computing are typically based on the Intel x86 architecture rather than the IBM 370 architecture, but that is a minor detail relative to the big picture.

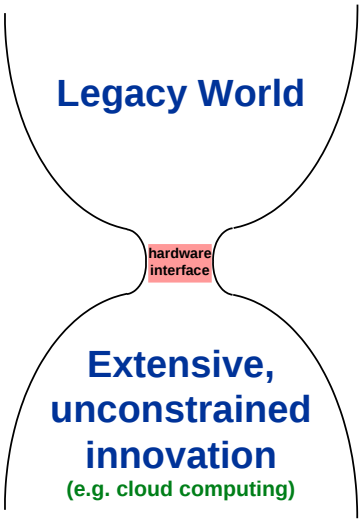


Figure 2: Legacy-compatible VM Ecosystem

for the Android platform [28]. While these alternatives (collectively referred to as *software virtualization*) have successful niches, they do not approach the VM abstraction (also referred to as *hardware virtualization*) in terms of longevity, widespread usage and real-world impact.

Why is hardware virtualization so much more successful? First, the interface presented by the VM abstraction 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 written in specific languages such as Java. In contrast, a VM is language-agnostic and OS-agnostic. In fact, a JVM can be part of the ecosystem supported by a VM. Hardware virtualization can thus subsume software virtualization. Second, a VM interface is *narrow* and *stable* relative to typical software interfaces. In combination, these two attributes ensure adequate return on investments in the layer below. By definition, a narrow interface imposes less constraints on the layer below and thus provides greater freedom for innovation. 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 maintenance effort. Pliability also leads to widening of narrow interfaces over time, because it is difficult to resist the temptation to extend an interface for the benefit of a key application. Over time, the burden of sustaining a wide interface constrains innovation below the interface.

The importance of the narrowness and stability of an interface can be seen in the contrasting fortunes of *process migration* and *VM migration,* which are essentially the same concept applied at different levels of abstraction. Process migration is an operating system capability that allows a running process to be paused, relocated to another machine, and continued there. It has been a research focus of many experimental operating systems built in the past 20 years. Examples include Demos [18], V [24], Mach [30], Sprite [11], Charlotte [7], and Condor [29]. These independent validation efforts have shown beyond reasonable doubt that process migration can indeed be implemented with acceptable efficiency. Yet, in spite of its research popularity, no operating system in widespread use today (proprietary or open source) supports process migration as a standard facility. The reason for this paradox is that a typical implementation of process migration involves such a wide interface that it is easily rendered incompatible by a modest external change such as an operating system upgrade. Long-term maintenance of machines with support for process migration involves too much effort relative to the benefits it provides. The same concept of pausing an executing entity, relocating it, and resuming execution can be applied to an entire VM rather than

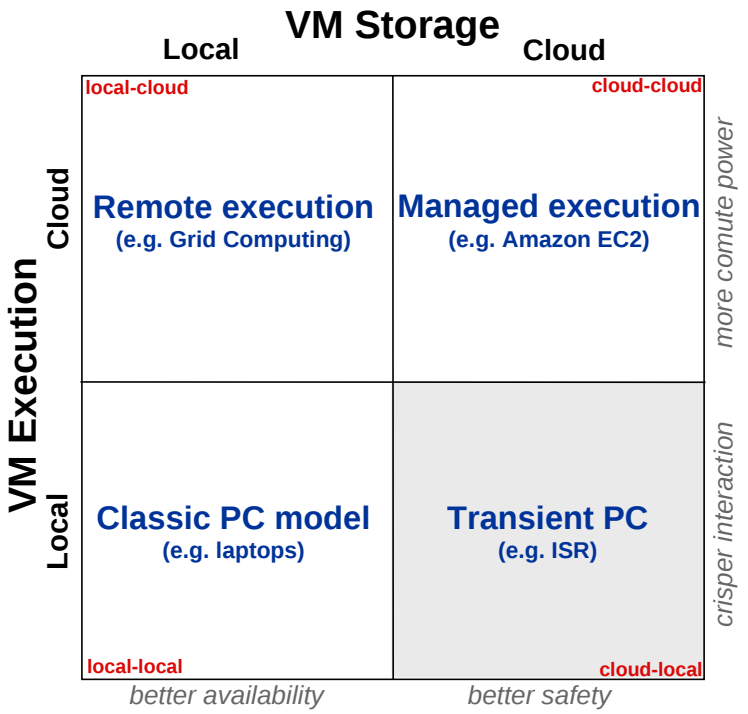


Figure 3: VM-based Cloud Computing Strategies

a single process. This is *VM migration,* a capability that is in widespread use in cloud computing systems today. Guest OS changes do not affect an implementation of VM migration. This insulation from change, embodied in the narrow and stable VM interface, is crucial to the real-world success of VM migration.

4 Transporting VMs across Space and Time

Modern implementations of the VM abstraction provide support for an executing VM to be *suspended.* This is conceptually similar to what happens when you close the lid of a laptop computer. An exact record of the execution state of the VM at suspend is captured and saved as files in the host file system. There are typically distinct representations of the VM's persistent state (i.e., its virtual disks) and its volatile state (i.e. memory, registers, etc.). These are referred to as the VM's *disk image* and *memory image* respectively. For brevity, we refer to a combination of the two as a *VM image.* Execution can be *resumed* from a VM image. This is equivalent to opening the lid of a suspended laptop, and seamlessly continuing work. If a VM is shut down, rather than suspended, there is no memory image saved. Resuming such a VM is equivalent to powering up hardware, resulting in a fresh boot of the guest operating system. An executable VM that is created from a VM image is called a *VM instance.* Some use cases may restrict the number of concurrent VM instances that can be created from a single VM image.

A VM image is static content that can transmitted across the Internet for sharing, or archived for later retrieval. If you have a VM image, resuming the suspended VM only requires a compatible VMM and host hardware. Everything else needed for perfect execution fidelity is contained within the VM image. The use of VMs thus transforms the difficult problem of collaborating with executable content into the more tractable problem of efficient transmission, storage and retrieval of VM images.

Separating image storage from instance execution leads to a rich design space for VM-based cloud computing. As Figure 3 shows, this design space has two dimensions. One dimension is the location of the definitive copy of the VM image (ignoring cache copies): local to the execution site, or in the cloud. The

other dimension is where a VM instance executes: locally (at the edges of the Internet) or within the cloud. This results in the four quadrants depicted in Figure 3:

- The local-local quadrant is equivalent to the classic unvirtualized model of desktops and laptops today.
- The cloud-cloud quadrant corresponds to situations where both VM image storage and execution occur within the cloud. This is the model supported by public clouds such as Amazon Web Services, and is often what is meant by the term "cloud computing."
- The local-cloud quadrant corresponds to situations where a local VM image is instantiated in a remote cloud, typically for use of more powerful computing resources or for proximity to large data sets. This is essentially the metaphor of *grid computing*. Use cases in which large data sets are shared by a community of researchers using this mechanism can be viewed as a form of collaboration across space.
- The cloud-local quadrant is of particular interest to the topic of this paper and is discussed in detail below. This quadrant corresponds to a VM image archived in the cloud being instantiated and executed at an edge node of the Internet.[3] We label this quadrant *Transient PC* because it evokes the vision of a transient or disposable computing instance that is brought into existence anywhere on the Internet, used and then discarded. If its final state is valuable, that could be archived as a new image in the cloud.

The Transient PC model is well suited to supporting collaboration across time. Just as we retrieve a `pdf` file today, view it on an edge device such as a desktop, laptop or smartphone and then discard the file, we can envision a VM corresponding to executable content being dynamically instantiated and executed on an edge device. There are many details to get right, but in principle this approach offers far better execution fidelity than any of the alternatives available today. This is because the entire transitive closure of dependencies (including the operating system, dynamically linked libraries, supporting applications, and so on) are archived along with the application of interest within the VM image.

Our work has confirmed the feasibility of using the Transient PC model for executing obsolete content. Figure 4 through Figure 7 show screenshots from executing instances of VM images that were created from the original install disks of long-obsolete software. These include the Windows 3.1 ecosystem on Intel x86 hardware, the Mac OS ecosystem on Motorola 68040 hardware, and the OpenStep ecosystem for x86 hardware. The Mac OS example involve two layers of hardware emulation. The first layer of emulation is for now-obsolete hardware (Motorola 68040). This emulator as well as the VM image it uses (created from the original install disks) are encapsulated within an outer VM image that is archived. Execution speed is acceptable in spite of two layers of hardware emulation because of the enormous improvement in processor performance that has occurred since the heyday of these obsolete ecosystems.

With additional machinery, the Transient PC model could also support near real-time collaboration across space between multiple users. This would consist of a VM instance executing at each user's site, along with mechanisms to propagate changes resulting from each user's interactions to the other VM instances. In effect, a collection of VM instances dispersed across the edges of the Internet are kept in sync as they interact with their users. This achieves the effect of the remote desktop approach mentioned in Section 2, but with much better execution fidelity since it is less sensitive to wide-area Internet latency. There is an extensive body of research on logging and replay of VMs that could be leveraged for this purpose [12, 13, 27]. In addition, there would need to be an appropriate model of concurrency control to ensure that the divergence of VM instances remains within acceptable bounds. The simplest concurrency control approach would be a token-passing scheme that allows at most one user at a time to be the "writer." More

[3]The private cloud of an enterprise can be viewed as "an edge node of the Internet" for this discussion.

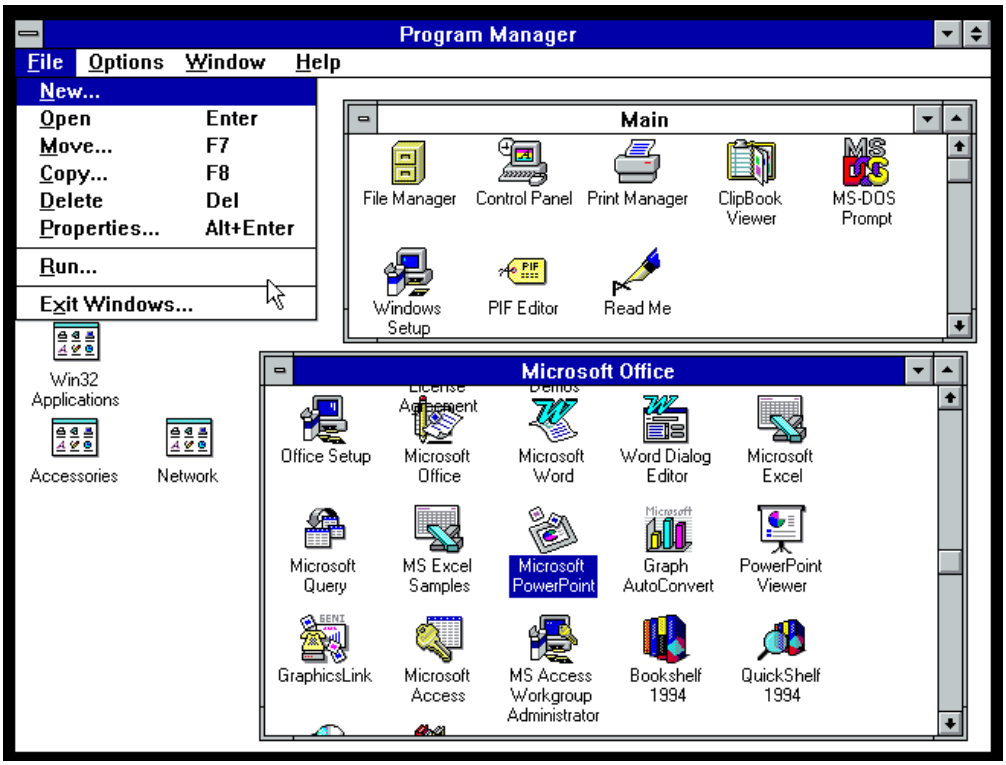


Figure 4: Windows 3.1 as Guest Operating System within a VM (screenshot captured on September 28, 2011)

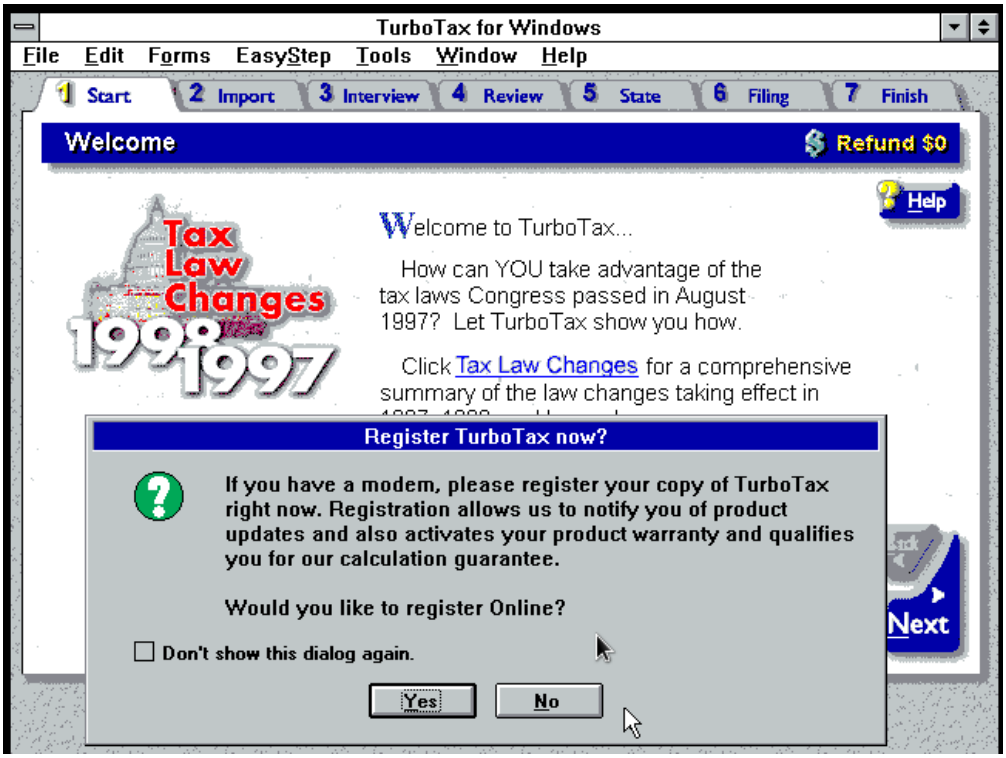


Figure 5: TurboTax 1997 Executing within Windows 3.1 Guest in a VM (screenshot captured on September 28, 2011)

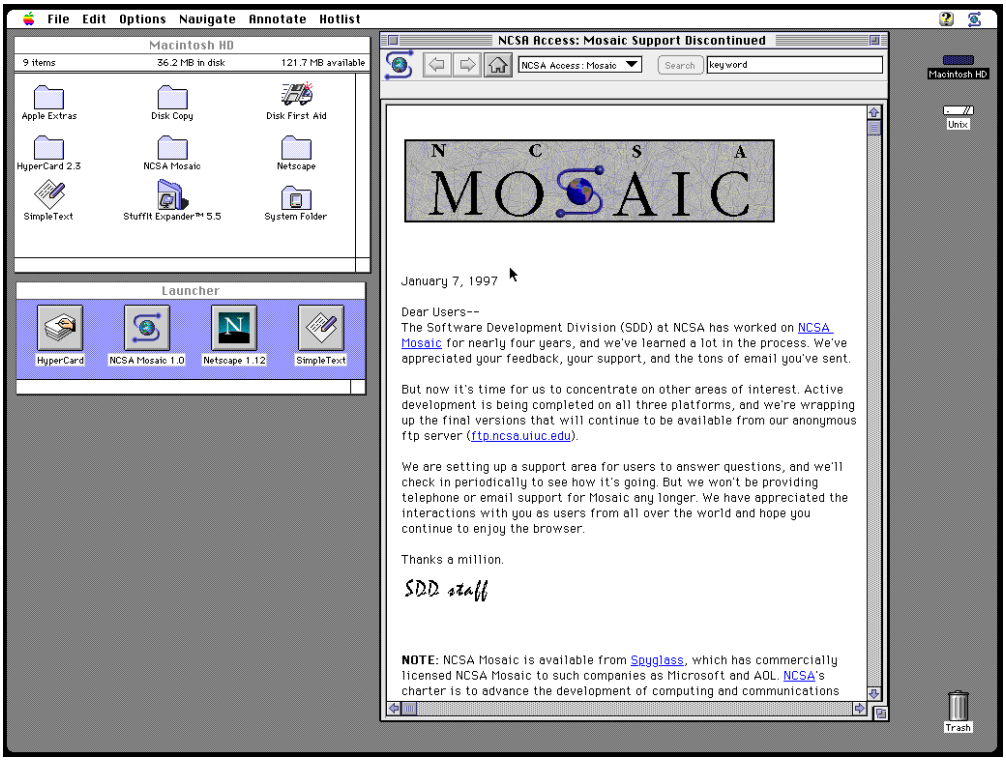


Figure 6: Mac OS for Motorola 68040 hardware with Mosaic 1.0 in a VM (screenshot captured on September 28, 2011)

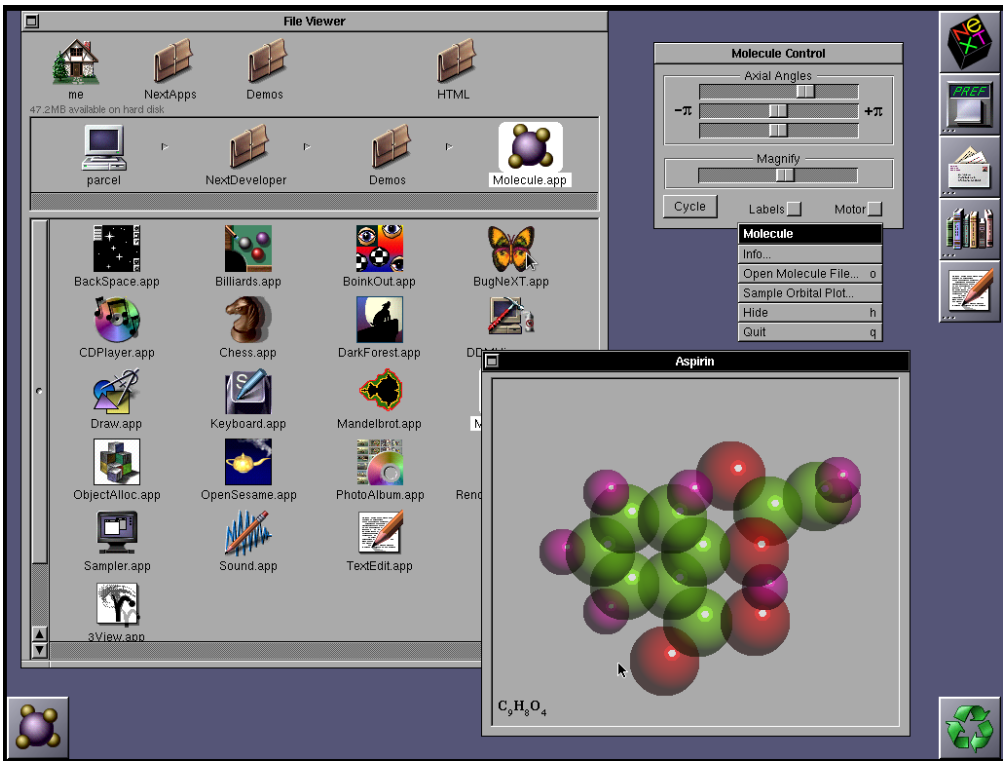


Figure 7: OpenStep for x86 Hardware in a VM (screenshot captured on September 28, 2011)

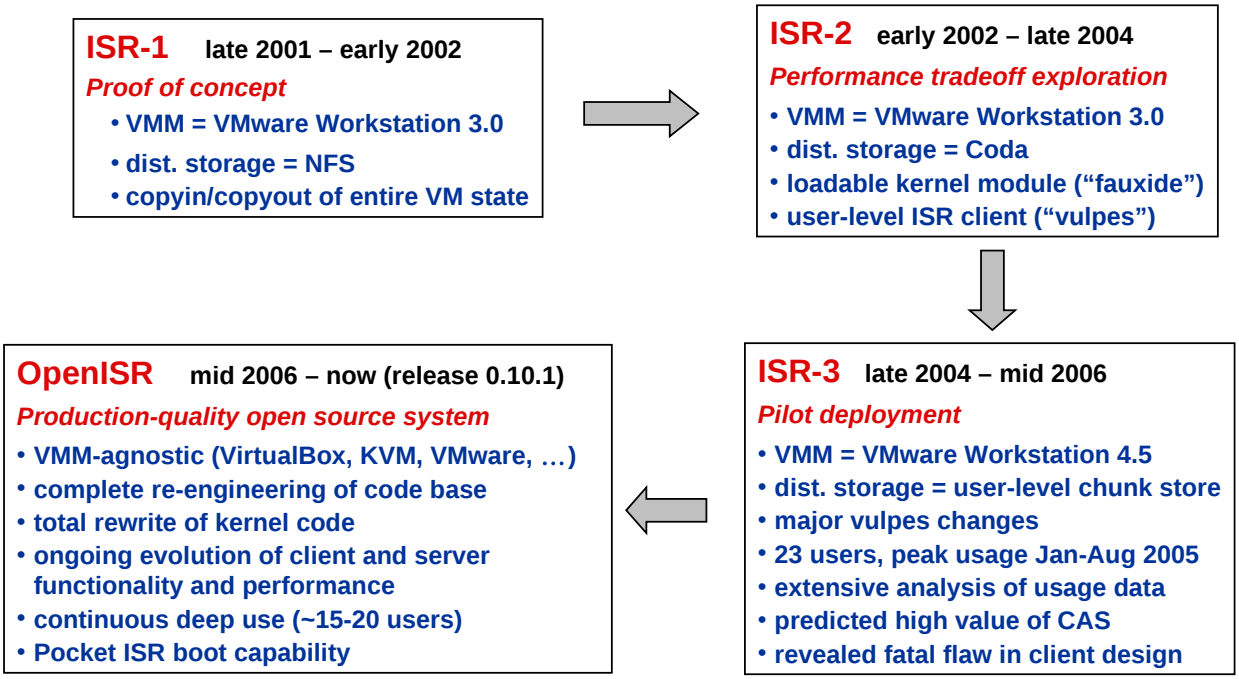


Figure 8: Evolution of the Internet Suspend/Resume® (ISR) System

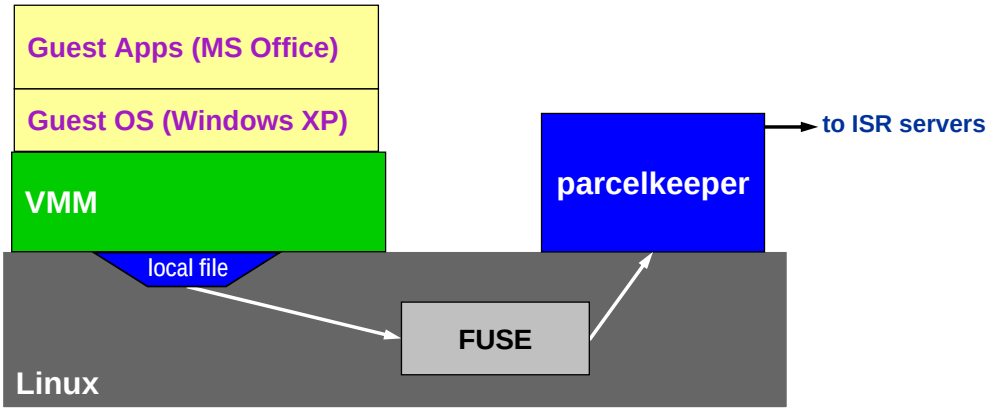


Figure 9: ISR Client Structure

sophisticated models of collaboration will require more complex concurrency control approaches. For example, multi-user graphics-intensive computer gaming at Internet scale may benefit from this approach if game-specific concurrency control can be efficiently supported.

The superior execution fidelity of VMs comes at a price. A typical VM image is many gigabytes in size, possibly tens of gigabytes. In contrast, a typical `pdf` document is tens of kilobytes to a few megabytes in size. While storage capacity in the cloud for VM images is of some concern, a more serious concern is their efficient transmission over the Internet. Even at 100 Mbps, full transfer of a 10 GB image will take at least 800 seconds (over 13 minutes). Over the wide-area Internet, with end-to-end bandwidths approaching 10 Mbps at well-connected sites, the transfer will take over 8000 seconds (over two hours). These are clearly unacceptably long periods for a user to wait before executable content can be viewed. Ideally, the startup delay experienced by a user at a well-connected Internet site should be no more than a few seconds to tens of seconds. While Internet bandwidth will continue to improve over time, it is also likely that VM images will grow larger.

The obvious solution is to adopt the approach of *streaming* used by video sites such as YouTube. This would allow a user to begin viewing content as soon as a modest prefix has been transferred. Unfortunately, the problem is more complex than video streaming because a VM instance is unlikely to access its VM image in simple linear order. It is the complex runtime behavior of a VM instance (including user interactions, data dependencies, temporal locality and spatial locality) that determines the reference pattern to its VM image. To address this problem, we have shown how *demand-paged execution* of a VM instance can be implemented in a VMM-agnostic manner. Using this approach resume latencies of tens of seconds are achievable today at well-connected Internet sites. This work has been done in the context of the *Internet Suspend/Resume®(ISR)* system [3, 22, 21]. Figure 8 summarizes the evolution of this system, while Figure 9 illustrates the key components of an ISR client.

5 The Olive Vision

We have shown that VMs can archive executable content over time periods of at least 10–15 years, and possibly much longer. To realize the full potential of this capability we envision *Olive,* a VM-based Internet ecosystem for executable content. At the heart of Olive is open-source software for contributing VM images, curating them in Internet repositories so that licensing and intellectual property constraints are enforced, and efficiently retrieving and executing them at a cloud provider or at an edge device (i.e., the cloud-cloud and cloud-local quadrants of Figure 3). The ISR mechanism for streamed execution of VM images (mentioned in Section 4) is a good starting point for this body of software, but much additional functionality has to be conceived, designed and implemented.

As shown in Figure 10, Olive is a concept that applies at multiple levels of the Internet. At the global or planetary level, we envision a relatively small number of archival sites (less than 100, more likely 1–10) where VM images are curated. Olive also has a presence at the level of a public cloud provider such as Amazon Web Services, or within the private cloud of an enterprise (most likely totaling a few hundred to a few thousand clouds). At this level, VM images may be cached in their entirety from global Olive repositories. They may also have customizations that are cloud-specific or enterprise-specific. Finally, Olive's presence at any edge device on the Internet enables ubiquitous streamed execution of VM images over wired and wireless networks at any time and place (totaling 100 million or more devices worldwide). For scalability, VM images may be streamed to an edge device from a cloud provider rather than directly from a global Olive repository.

The ability to customize VM images within a cloud is an important Olive capability because it helps VM image contributors to honor software licensing constraints. Executable software content created by one person almost always depends on other software content. Take for example someone who wants to

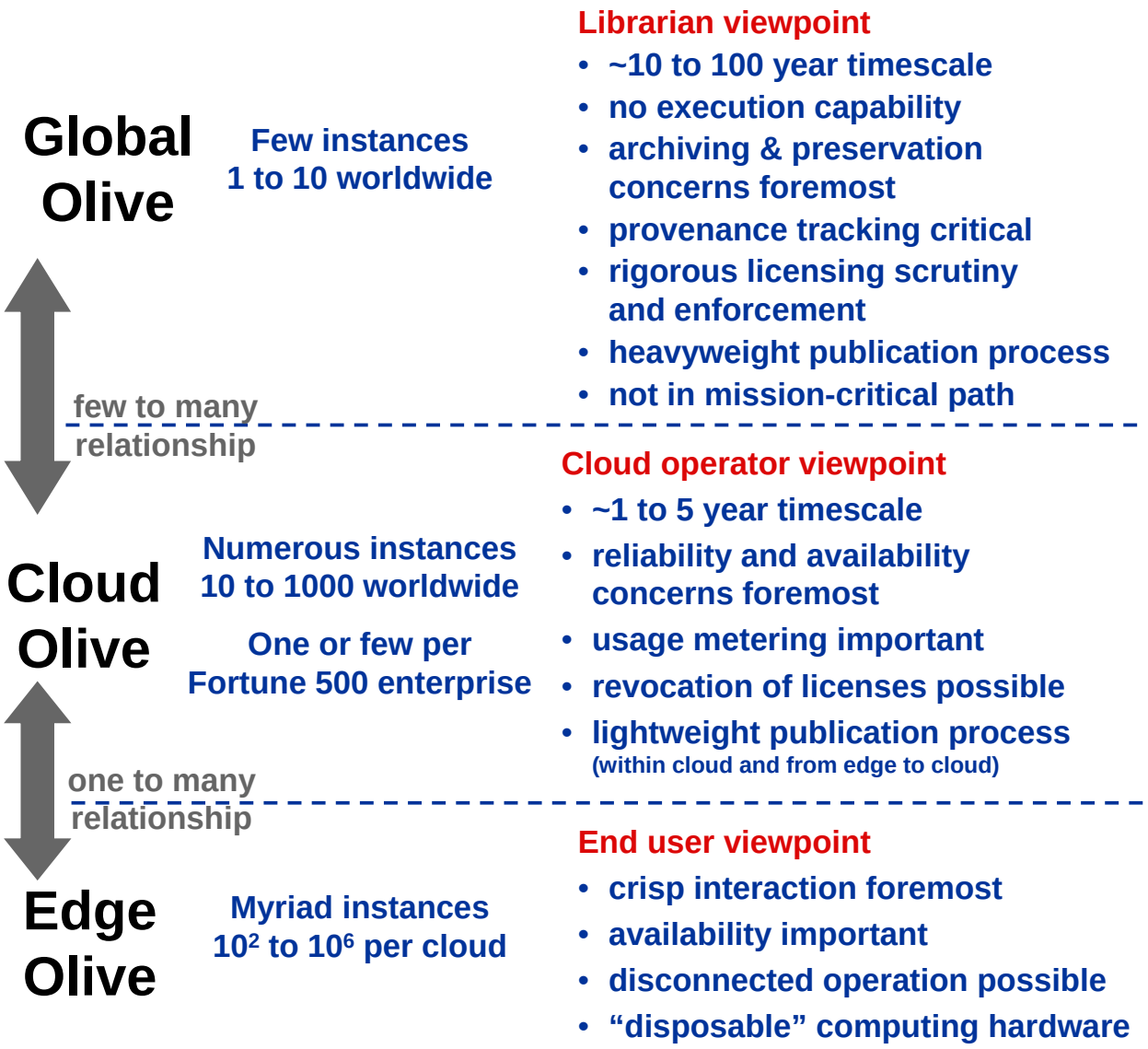


Figure 10: Olive as a Concept at Multiple Levels

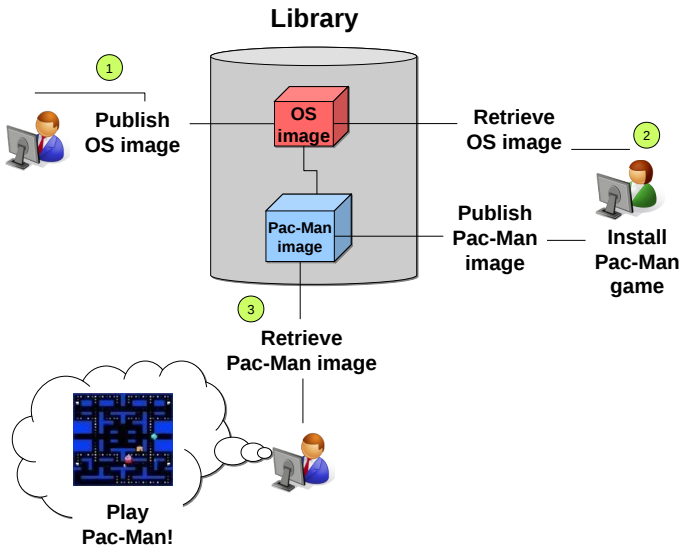
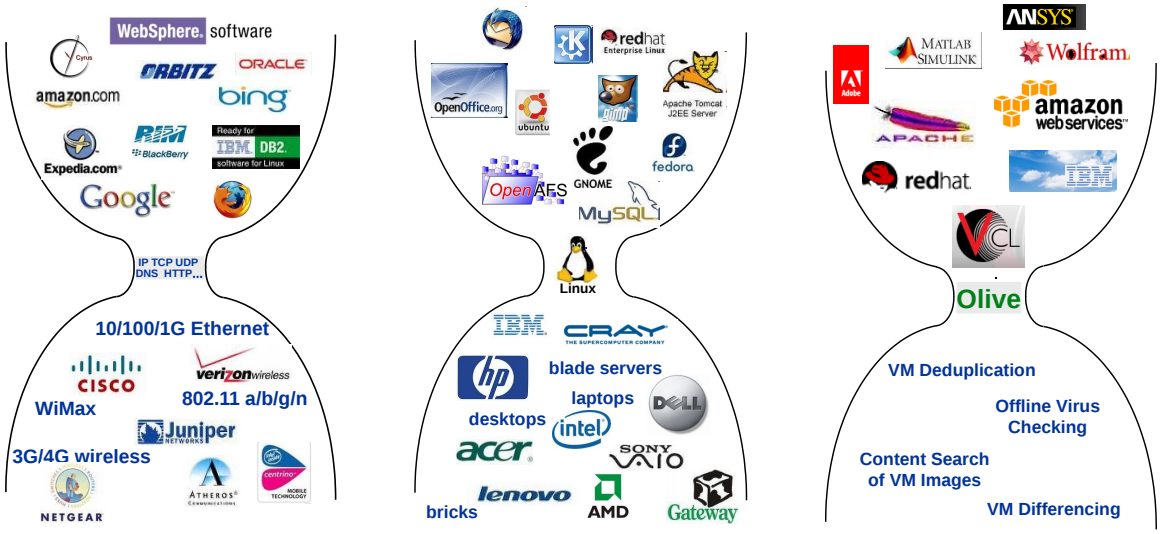


Figure 11: Extending an Olive VM Image

contribute the popular computer game Pac-Man to a global Olive repository. This person may have the rights to contribute the Pac-Man application, but not the rights to the operating system and other software that must exist on the VM inside which Pac-Man has to run. If the contribution to Olive included those other software components, it would require negotiation of re-distribution rights on all those other components with their corresponding owners. This is not an issue with a totally open source software stack, but it can be a laborious and expensive task when proprietary software is involved. This legal hurdle and its associated litigation and business risks could discourage many from making valuable Olive contributions.

Olive solves this problem by allowing one VM to extend another VM, thereby creating a new VM. Figure 11 illustrates one way this can work. In step 1, a person or organization that wants to contribute an operating system to Olive, does so as a VM that includes a fully installed and tested image of that operating system. A different person or organization can then retrieve this VM, start it on their own laptop, and install the Pac-Man application. The resulting VM, which includes both the operating system and the Pac-Man application, can then be published back into Olive. Olive technology will recognize that this new VM is an extension of a virtual machine that already exists in its library. It will automatically compute the delta content between the two (which is the contribution from the Pac-Man application), and store this delta internally. Thus, the Pac-Man contributor only contributes bits that he owns to Olive and does not violate any licensing restrictions. This process is reversed during the customization step within a cloud: the Pac-Man delta is applied to the original VM to produce a merged VM.

Olive will be an ecosystem built around open standards and interfaces rather than a closed, proprietary system. The vibrant Internet and Linux ecosystems shown in Figures 12(a) and 12(b) are testimonials to the power of open standards. In both cases, a relatively small collection of open components at the waistline sustains large business investments above and below that waistline. Figures 12(c) illustrates the hypothetical Olive ecosystem of the future. At the waistline are a small set of open components such as ISR. Above are publishers of open or proprietary executable content. Below are open or proprietary tools and mechanisms. We believe that such an open ecosystem has the potential to be far more influential and viable than a proprietary ecosystem.



(a) Internet Today (b) Linux Today (c) Future Olive

Figure 12: The Power of Open Ecosystems

6 Olive and Digital Libraries

Historically across the disciplines, academic libraries have maintained the scholarly record, which fixes credit for innovation, invention, and progress. For the stone tablet, papyrus, vellum, and paper environments, successful retention has been challenging but attainable. The realm of static digital resources has proven more challenging [16], but standards and best practices have led several libraries through successful migrations of large corpuses. Libraries are also the logical stewards for executable content as for other scholarly content. Academic libraries have universally accepted their faculty's research and publication products, which may range from cooking, to chemistry, to computing. The last now complain that they cannot fully appreciate and understand work if it is just represented by object code divorced from source code. Computer scientists need to see what a program does when it runs to be fully informed and enriched by their tradition. As Olive is realized, libraries, working with their computing colleagues, will be able to commit to the long term preservation of diverse forms of executable content. Examples include tutoring systems (such as Loop Tutor, Hypertutor, and iCarnegie), games (such as Rogue, MUDs, Doom, and Duke Nukem), databases (such as Clueset), and obsolete versions of major commercial operating systems and applications.

Trying to maintain the equipment to run these older works is not an elegant preservation solution. Many of the first choices of works to be archived may, of necessity, be those with less commercial value. The JSTOR database has demonstrated that rights owners can be persuaded to give permission for the use of less valuable content (journal articles over five years old). Nevertheless, intellectual property restrictions will be a significant challenge. Another significant challenge will be sustaining the archive for academic products long term. Many models exist: JSTOR (subscription), the dark archives Portico and LOCKSS (membership), and NIH's PubMedCentral and Bibliothque Nationale de France's DeepArc (government funding). The Million Books Project model (multinational governments, subscription, and university computer science department and library support) has proven to be less effective [23].

7 Conclusion

Rich shared context facilitates human collaboration. Face to face interaction is the gold standard for shared context, since it supports the full range of visual, aural and tactile real-time interactions involved in conveying procedural and declarative knowledge. Even in the 21^{st} century, people travel to meetings, workshops and conferences for this highest quality of shared context.

Extending collaboration across space and time to approach this gold standard has been a human quest since the dawn of civilization. The invention of writing, and the emergence of libraries profoundly changed collaboration across time. Over centuries, this quest has led to other transformative inventions such as the printing press, telephony, the motion picture, television, the Internet, and social networks such as Facebook. Yet, in spite of these impressive advances, the shared context supported by them has always been static in nature. Today's Flickr images and YouTube videos may only bear a distant relationship to writing on clay tablets, but they both represent static content from the viewpoint of archiving. Sharing executable content has been theoretically possible since the invention of the computer. This would enhance and enrich collaborations, bringing them closer to the gold standard. But only now have the foundational technologies for this capability matured to the point where we can build upon them.

Procedural artifacts ranging from scientific simulation models to interactive games play an increasingly important role in our lives. 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 now. As we have shown in this paper, there is now a convergence of key technologies needed for such scenarios. Foremost among these are VM technology, cloud computing and high bandwidth wide-area networks. From these roots, the Olive vision emerges as a natural next step for supporting collaborations across space and time.

Acknowledgements

We acknowledge the extensive contributions of Benjamin Gilbert, who has been the primary contributor to ISR since 2006. Earlier contributors to ISR include Michael Kozuch, Casey Helfrich, Dave O'Hallaron, Partho Nath, and Matt Toups. We also acknowledge the many people who contributed to the Olive vision presented here: Benjamin Gilbert, Jan Harkes, Glenn Ammons, Catherine Zhang, Mladen Vouk, Peng Ning and many other contributors at Carnegie Mellon, IBM, and North Carolina State University, as well as the participants of the Olive workshop held at IBM in April 2011. We thank Dan Ryan for his contributions to this paper, and David Baisley for providing us with the original install disks for the operating systems shown in Figure 4 through Figure 7. This research was supported by the National Science Foundation (NSF) under grant numbers CNS-0833882 and CNS-0509004, an IBM Open Collaborative Research grant, and Intel. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily represent the views of the NSF, IBM, Intel, or Carnegie Mellon University. Internet Suspend/Resume and OpenISR are registered trademarks of Carnegie Mellon University.

References

[1] Cisco Webex. http://www.webex.com/, September 2011.

[2] GoToMyPC: Access Your Mac or PC from Anywhere. http://www.gotomypc.com/remote_access/remote_access, September 2011.

[3] Internet Suspend/Resume. http://isr.cmu.edu/, September 2011.

[4] Remote Desktop Protocol. http://en.wikipedia.org/wiki/Remote_Desktop_Protocol, September 2011.

[5] VM (operating system). http://en.wikipedia.org/wiki/VM(operatingsystem), September 2011.

[6] AMAZON. Amazon Web Services. http://aws.amazon.com/, September 2011.

[7] ARTSY, Y., AND FINKEL, R. Designing a Process Migration Facility: The Charlotte Experience. *IEEE Computer 22*, 9 (1989).

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

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

[10] CREASY, R. The Origin of the VM/370 Timesharing System. *IBM Journal of Research and Development 25*, 5 (1981).

[11] DOUGLIS, F., AND OUSTERHOUT, J. Transparent Process Migration: Design Alternatives and the Sprite Implementation. *Software Practice and Experience 21*, 8 (1991).

[12] DUNLAP, G. W., KING, S. T., CINAR, S., BASRAI, M., AND CHEN, P. M. ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay. In *Proceedings of the 2002 Symposium on Operating Systems Design and Implementation (OSDI)* (December 2002).

[13] KING, S. T., DUNLAP, G. W., AND CHEN, P. M. Debugging operating systems with time-traveling virtual machines. In *Proceedings of the 2005 Annual USENIX Technical Conference* (April 2005).

[14] LINDHOLM, T., AND YELLIN, F. *The Java Virtual Machine Specification (2nd Edition)*. Prentice Hall, 1999.

[15] MATTHEWS, B., SHAON, A., BICARREGUIL, J., AND JONES, C. A Framework for Software Preservation. *The International Journal of Digital Curation 5*, 1 (June 2010).

[16] MCDONOUGH, J. ET AL. Preserving Virtual Worlds. http://hdl.handle.net/2142/17097, September 2010. Also see "Preserving Virtual Worlds" project website: http://pvw.illinois.edu/pvw/.

[17] Oxford Dictionaries. http://oxforddictionaries.com, September 2011.

[18] POWELL, M., AND MILLER, B. Process Migration in DEMOS/MP. In *Proceedings of the 9th ACM Symposium on Operating Systems Principles* (Bretton Woods, NH, Oct. 1983).

[19] RICHARDSON, T., STAFFORD-FRASER, Q., WOOD, K. R., AND HOPPER, A. Virtual Network Computing. *IEEE Internet Computing 2*, 1 (Jan/Feb 1998).

[20] SATYANARAYANAN, M., BAHL, V., CACERES, R., AND DAVIES, N. The Case for VM-based Cloudlets in Mobile Computing. *IEEE Pervasive Computing 8*, 4 (Oct-Dec 2009).

[21] SATYANARAYANAN, M., GILBERT, B., TOUPS, M., TOLIA, N., SURIE, A., O'HALLARON, D. R., WOLBACH, A., HARKES, J., PERRIG, A., FARBER, D. J., KOZUCH, M. A., HELFRICH, C. J., NATH, P., AND LAGAR-CAVILLA, H. A. Pervasive Personal Computing in an Internet Suspend/Resume System. *IEEE Internet Computing 11*, 2 (2007).

[22] SATYANARAYANAN, M., KOZUCH, M., HELFRICH, C., AND O'HALLARON, D. R. Towards Seamless Mobility on Pervasive Hardware. *Pervasive and Mobile Computing 1*, 2 (June 2005).

[23] ST. CLAIR, G. Challenges in Sustaining the Million Book Project, a project supported by the National Science Foundation. In "The Selected Works of Gloriana St. Clair" at `http://works.bepress.com/gloriana\_stclair/19`, 2010.

[24] THEIMER, M., LANTZ, K., AND CHERITON, D. Preemptable Remote Execution Facilities for the V-System. In *Proceedings of the 10th Symposium on Operating System Principles* (Orcas Island, WA, December 1985).

[25] TOLIA, N., ANDERSEN, D., AND SATYANARAYANAN, M. The Seductive Appeal of Thin Clients. Tech. Rep. CMU-CS-05-151, School of Computer Science, Carnegie Mellon University, March 2005.

[26] TOLIA, N., ANDERSEN, D., AND SATYANARAYANAN, M. Quantifying Interactive Experience on Thin Clients. *IEEE Computer 39*, 3 (March 2006).

[27] VEERARAGHAVAN, K., LEE, D., WESTER, B., OUYANG, J., CHEN, P. M., FLINN, J., AND NARAYANASAMY, S. DoublePlay: Parallelizing sequential logging and replay. In *Proceedings of the 2011 International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (March 2011).

[28] WIKIPEDIA. Dalvik (software). `http://en.wikipedia.org/wiki/Dalvik\_(software)`, September 2011.

[29] ZANDY, V., MILLER, B., AND LIVNY, M. Process Hijacking. In *8th International Symposium on High Performance Distributed Computing* (Redondo Beach, CA, August 1999).

[30] ZAYAS, E. Attacking the Process Migration Bottleneck. In *Proceedings of the 11th ACM Symposium on Operating System Principles* (Austin, TX, November 1987).