Interactive Resource-Intensive Applications Made Easy

H. Andrés Lagar-Cavilla[1], Niraj Tolia[2], Eyal de Lara[1], M. Satyanarayanan[2], and David O'Hallaron[2]

[1] University of Toronto
[2] Carnegie Mellon University

Abstract. Snowbird is a middleware system based on virtual machine (VM) technology that simplifies the development and deployment of *bimodal* applications. Such applications alternate between phases with heavy computational-resource needs and phases rich in user interaction. Examples include digital animation, as well as scientific, medical, and engineering diagnostic and design tools. Traditionally, these applications have been manually partitioned into distributed components to take advantage of remote computational resources, while still providing low-latency user interaction. Instead, Snowbird lets developers design their applications as monolithic units within a VM, and automatically migrates the application to the optimal execution site to achieve short completion time and crisp interactive performance. Snowbird does not require that applications be written in a specific language, or use specific libraries, and it can be used with existing applications, including closed-source ones, without requiring recompilation or relinking. Snowbird achieves these goals by augmenting VM migration with an interaction-aware migration manager, support for graphics hardware acceleration, and a wide-area peer-to-peer storage system. Experiments conducted with a number of real-world applications, including commercial closed-source tools, show that applications running under Snowbird come within 4% of optimal compute time, and provide crisp interactive performance that is comparable to native local execution.

Keywords: Bimodal Applications, Migration, Virtual Machines, Thin Clients, Interactive Response, Variable Thickness.

1 Introduction

A growing number of applications in many domains combine sophisticated algorithms and raw computational power with the deep knowledge, experience and intuition of a human expert. Examples of such applications can be found in simulation and visualization of phenomena in scientific computing, digital animation, computer-aided design in engineering, protein modeling for drug discovery in the pharmaceutical industry, and computer-aided diagnosis in medicine. These *bimodal* applications alternate between resource-intensive *crunch phases* that involve little interaction, and *cognitive phases* that are intensely interactive. During the crunch phase, short completion time is the primary performance goal, and computing resources are the critical constraints. During the cognitive phase, crisp interactive response is the primary performance goal, and user attention is the critical constraint.

Optimizing both phases is important for a good user experience, but achieving this end is complicated by the large disparity in the performance goals and bottlenecks of the two phases. Today, developers manually split a bimodal application into a distributed set of components [1–4]. Crunch phase components are executed on compute servers or server farms. Distant data servers are sometimes used because datasets are too large to cache or mirror locally, or are constrained by organizational or regulatory policies that forbid caching or mirroring. In contrast, cognitive phase components are executed locally where they can take advantage of local graphics acceleration hardware. Unfortunately, this approach requires developers to manage communication and coordination between application components, and to be aware at all times of whether a particular component will be executed locally or remotely. This adds software complexity above and beyond the intrinsic complexity of the application being developed, and hence slows the emergence of new bimodal applications.

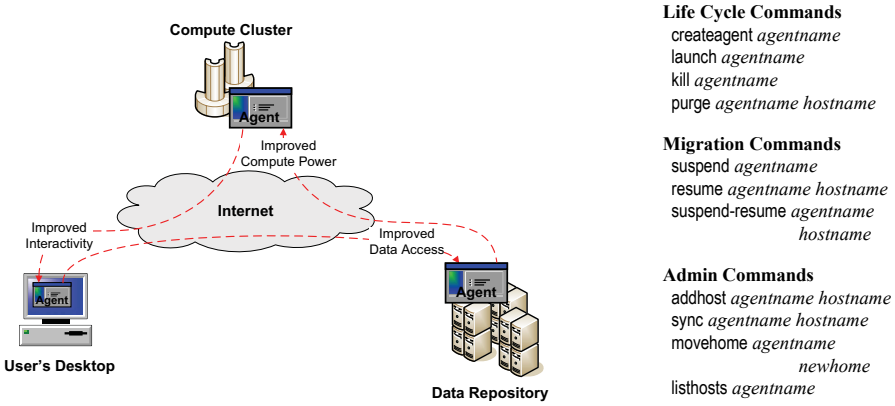
This paper introduces Snowbird, middleware based on virtual machine (VM) technology that simplifies the development and deployment of bimodal applications. Snowbird masks the complexity of creating a bimodal application by wrapping the application, including all its executables, scripts, configuration files, dynamically linkable libraries, and operating system, into a migratable VM. During execution, Snowbird automatically detects phase transitions in the application and migrates the VM containing its complex web of dependence to the optimal execution site. Snowbird does not require applications to be written in a specific language, nor to be built using specific libraries. Existing closed-source applications can use Snowbird without recoding, recompilation, or relinking.

Snowbird extends existing VM technology with three mechanisms: an interaction-aware migration manager that triggers automatic migration; support for graphics hardware acceleration; and a peer-to-peer storage subsystem for efficient sharing of persistent VM state at Internet scale. Experiments conducted with a number of real-world applications, including commercial closed-source tools such as the Maya 3D graphics animation package, show that applications running under Snowbird come within 4% of optimal crunch completion times, while exhibiting crisp interactive performance that is comparable to native local execution.

From a more abstract perspective, Snowbird can be viewed as a tool that provides seamless transitions between thick and thin client modes of execution. It has long been known that the strengths of thick and thin clients complement each other. Thin clients are attractive in CPU-intensive and data-intensive situations because application execution can occur on remote compute servers close to large datasets. Unfortunately, high network latency and jitter between the application execution site and the user site can lead to poor interactive performance. Thick clients offer a much better user experience in that situation. By transparently morphing an application between thick and thin client modes of execution, Snowbird gives a user the best of both worlds.

2 Design and Implementation

The dominant influence on the design of Snowbird was our desire to simplify the creation and deployment of bimodal applications without imposing onerous constraints on developers. The use of VM technology is the key to achieving this goal. It allows a



(a) Agent Migration Concept (b) Snowbird Commands

Fig. 1: Snowbird Overview

developer to focus on the creation of a single monolithic entity rather than the more difficult programming task of creating a distributed system. This monolithic entity, called an *agent,* is the migratable embodiment of an application that transparently and seamlessly relocates itself to achieve optimal performance. At run time, Snowbird automatically detects phase transitions and migrates the agent to the optimal execution site. The example illustrated in Figure 1(a) shows an agent that starts at the user's desktop to provide good interactive response during a cognitive phase. It then migrates to several remote sites, where it leverages the superior compute power of a shared-memory multiprocessor and improved I/O performance from proximity to a large dataset. The agent then returns to the desktop for the next cognitive phase.

The logical encapsulation provided by an agent eases the complexity of developing and deploying a bimodal application. Simplicity is reinforced by the fact that all the paraphernalia associated with a large application (such as other processes, dynamically linked libraries, and specific OS features upon which an application relies) is atomically moved with the same containing agent. Hence no application-specific code has to be pre-installed in order to run on a site. An agent only requires SSH access credentials to execute on a Snowbird-enabled site. The SSH credentials are also used to encrypt all communications. Note that an agent can be a tool chain composed of several processes executing simultaneously or sequentially in a pipeline fashion.

Snowbird's use of VM technology offers three significant advantages over existing approaches to code mobility. First, applications do not have to be written in a specific language, to be built using specific libraries, or to run on a particular OS. Second, legacy applications do not have to be modified, recompiled, or relinked to use Snowbird. This greatly simplifies real-world deployments that use proprietary rather than open-source applications. Third, migration is transparent and seamless to the user, beyond the obviously desirable effects of improved interactive or computational performance.

A second factor influencing Snowbird's design is our desire to support applications at an Internet scale, particularly those using remote datasets over WAN links. It is end-to-end latency, not bandwidth, that is the greater challenge in this context. Table 1 shows recent round-trip time (RTT) values for a representative sample of Internet2 sites [5]. The theoretical minimum RTT values imposed by speed-of-light propagation, shown in the last column c, are already problematic. Unfortunately, technologies such as firewalls

End Points	RTTs (ms)			
	Min	Mean	Max	c
Berkeley – Canberra	174.0	174.7	176.0	79.9
Berkeley – New York	85.0	85.0	85.0	27.4
Berkeley – Trondheim	197.0	197.0	197.0	55.6
Pittsburgh – Ottawa	44.0	44.1	62.0	4.3
Pittsburgh – Hong-Kong	217.0	223.1	393.0	85.9
Pittsburgh – Dublin	115.0	115.7	116.0	42.0
Pittsburgh – Seattle	83.0	83.9	84.0	22.9

Table 1: Internet2 Round Trip Times

and overlay networks further exacerbate the problem, causing the minimum observed RTT values to far exceed the substantial propagation delays. Although bandwidths will continue to improve over time, RTT is unlikely to improve dramatically. These performance trends align well with the design of Snowbird; the critical resource in VM migration is network bandwidth, while the critical resource for crisp interaction is RTT.

A third factor influencing Snowbird's design is the peer-to-peer (P2P) relationship between migration sites that is implicit in Figure 1(a). Since Snowbird can migrate to any Internet site for which it possesses SSH credentials, there is no notion of clients or servers. Solely for purposes of system administration, Snowbird associates a *home* host with each agent. This is typically a user's desktop machine or so some other nearby computer where the user spends most of her time interacting with the agent. The home host acts as the authoritative machine on which the commands shown in Figure 1(b) are issued. The command line interface for Snowbird includes commands for managing an agent's life cycle, for controlling agent migration, and for system administration. Migration control commands are typically used by the migration manager described in Section 2.2. However, they are available for explicit user control, if desired.

Sections 2.1 to 2.4 present more details on four specific aspects of Snowbird. Section 2.1 expands upon our use of VM technology. Section 2.2 describes the interaction-aware migration manager. Section 2.3 describes the use of hardware-accelerated graphics by VM applications. Section 2.4 presents Snowbird's wide-area peer-to-peer storage subsystem for sharing persistent VM state.

2.1 Choice of VM Technology

The current version of Snowbird is based on the Xen 3.0.1 VMM. We chose Xen because its open-source nature makes it attractive for experimentation. However, our design is sufficiently modular that using a different VMM such as VMware Workstation will only require modest changes.

Snowbird uses VM migration [6, 7] to dynamically relocate the agent from a source to a target host. To migrate an agent, its VM is first suspended on the source. The suspended VM image, typically a few hundred MBs of metadata and serialized memory contents, is then transferred to the target, where VM execution is resumed. Snowbird uses *live-migration* [8] to allow a user to continue interacting with the application during agent relocation. This mechanism makes migration appear seamless, by iteratively prefetching the VM's memory to the target while the VM continues to execute on the source host. When the amount of prefetched VM memory reaches a critical threshold, a brief pause is sufficient to transfer control.

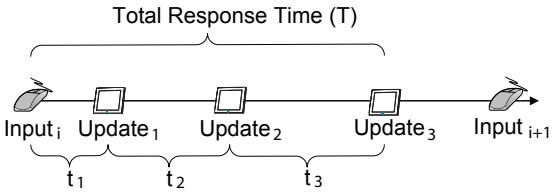


Fig. 2: Interaction Intensity and Smoothness

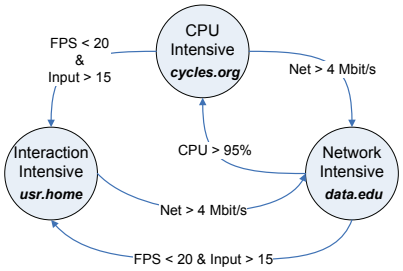


Fig. 3: Example Partial FSM

Modern VMMs allow the creation of VMs with multiple virtual CPUs regardless of the underlying number of available physical cores. Thus, when migrating from uniprocessor to multiprocessor hosts, Snowbird agents are able to transparently leverage the increased computing power by configuring their containing VMs as SMPs.

2.2 Interaction-Aware Migration Manager

While users can explicitly control migration decisions using the commands in Figure 1(b), Snowbird provides system-controlled agent relocation as one of its key features. In other words, the decision to migrate, the choice of migration site, and the collection of information upon which to base these decisions can all happen under the covers in a manner that is transparent to the user and to the agent.

A key feature of Snowbird is that it accounts for the quality of the application's interactive response when making its migration decisions. This is in stark contrast to the large body of related work on automated process migration policies [9], which concentrates on computationally-intensive applications devoid of interactions. Snowbird uses an *interaction-aware migration manager* module that bases its decisions on three sources: *interactivity sensors* that extract relevant data from the Snowbird user interface; *performance sensors* that extract their data from the VMM; and *migration profiles* that express the migration policy as transitions of a finite state machine triggered by sensor readings. Snowbird's clean separation between policy and mechanism simplifies the use of different profiles and sensors.

Interactivity sensors The interaction sensor is built into Snowbird's agent graphical user interface, described in the next section. As shown in Figure 2, the interaction sensor collects a stream of time-stamped events corresponding to keyboard/mouse inputs and screen updates. The intensity of the user's interactive demand and the quality of the agent's response can both be inferred from this stream.

Our measure of *interaction intensity* is the number of input events per unit of time. Our measure of *interactive response quality* is the number of frames per second triggered by an input event. This metric can be derived by assuming that all screen updates are causally related to the most recent input event. The frames per second (FPS) triggered by that input event is thus the number of related screen updates divided by the time from the event to the last of those updates. The FPS metric reflects both the smoothness and the swiftness of an interactive response. Remote interaction usually relies on non-work-conserving thin-client algorithms such as VNC [10] that under

adverse network conditions skip frames to "catch up" with the output. Skipping frames in this manner results in jerky on-screen tracking of mouse and keyboard inputs that can be annoying and distracting. For work-conserving thin-client algorithms like X, a low FPS rating means that the same amount of screen updates happened in more time, resulting instead in a sluggish response. We thus quantify the quality of the interactive response of an event window as the average FPS yielded by all the inputs in that window. High interaction intensity combined with a low-quality response is the cue used by the migration manager to trigger a remote-to-local transition.

Performance Sensors Snowbird provides performance sensors for *CPU utilization*, and *network utilization*. These sensors periodically poll the VMM for an agent's share of CPU time, and the number of bytes transmitted on its network interfaces, respectively. The poll interval is configurable with a default value of one second.

Migration Profiles A migration profile defines a finite state machine (FSM) that is used to model the agent's behavior. As shown in Figure 3, each state in this machine characterizes a particular level of resource demand and/or interaction. Profile rules define when and how sensor readings should trigger state transitions. The profile also specifies the amount of past sensor information that should be averaged to evaluate the rules, which defaults to ten seconds. Each state defines an optimal execution site. The mapping of application profile-defined FSM states to hosts is dependent on the infrastructure available to each particular user. While the figure exemplifies the typical FSM derived from the three sensors we implemented, profile writers are free to generate more complex FSMs using more sensors and states.

Profile creation involves a characterization of an agent's resource usage and may be done by application developers or by third-parties such as user groups, administrators, or technically adept users. In the absence of an application-specific profile, the migration manager uses a generic profile that identifies typical crunch and cognitive phases. The default profile is shown in Figure 3; most of its values are fairly intuitive and conservative. We use the long-established 20 FPS threshold [11] to trigger interactive response-based migrations. The input threshold of 15 inputs per window of ten seconds is derived from observations of the average input event generation rate in our experiments. We were able to use this generic application profile for all the experiments described in Section 3.5.

We plan to augment the Snowbird migration manager with many of the features developed by the process migration community in the scope of automated migration, such as selecting destination sites based on their current load [9]. One relevant concern in our environment is the handling of applications with overlapping crunch and cognitive phases, that could compromise the agent's stability by "thrashing" between the two states. The straightforward solution we have implemented is to specify a priority favoring interactive performance when conflicting migration rules are simultaneously triggered. Another solution would be to invoke hysteresis mechanisms [12] to prevent the migration manager from adopting this erratic behavior.

2.3 Hardware-Accelerated Graphical Interface

The graphical user interface for an agent has to comply with two requirements. First, a user should be able to interact seamlessly with an agent, regardless of its current

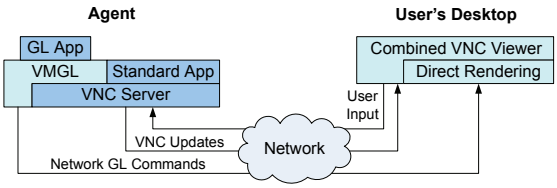


Fig. 4: 3D Support in Snowbird

location. Second, many of the applications targeted by Snowbird (such as scientific visualization and digital animation), require the use of 3D graphics acceleration hardware, a feature absent from most virtualized execution environments.

Snowbird uses VMGL [13] to meet these requirements. VMGL includes an enhanced thin client interface based on VNC [10], and provides agents with access to 3D graphics hardware acceleration. When the agent is running on a remote host, the thin client protocol is used to communicate screen updates and user inputs (i.e., keystrokes and mouse) over the network. When the agent runs on the user's desktop, the network becomes a loopback connection. Interaction is never interrupted during agent relocation because network connections persist through live-migrations: for relocations within the same L2 subnet, a gratuitous ARP-reply binds the agent's IP address to the new physical host. Relocations across subnets are supported with VPN tunnels or L2 proxys like VNETs [14].

VMGL provides applications running in a VM access to 3D graphics hardware acceleration by virtualizing the OpenGL API. This cross-platform API for 3D applications is supported by all major graphics hardware vendors. We use library preloading to masquerade as the system's native GL driver and intercept all GL calls made by an application. GL primitives are then forwarded over the network, using the WireGL protocol [15], to a remote rendering module where they are rendered directly by 3D graphics acceleration hardware. Although this setup allows complete flexibility, we expect the rendering module to execute in the user desktop's administrative VM, physically co-located with the agent VM during cognitive phases.

Figure 4 shows how we adapted VMGL for use in Snowbird. GL primitives bypass the VNC server and are rendered using 3D hardware on the user's desktop. Updates from non-3D APIs (e.g. Xlib) used by standard applications are rendered by the VNC server on its virtual framebuffer and shipped to the viewer. A modified VNC viewer composes both streams and offers a combined image to the user. Input events are handled entirely by the thin client protocol. Similar mechanisms can be used to support the Direct3D rendering API for Windows VMs [16].

2.4 The WANDisk Storage System

VM migration mechanisms only transfer memory and processor state; they do not transfer VM disk state, which is typically one to three orders of magnitude larger (many GBs). Therefore, each VM disk operation after migration usually involves network access to the source host. While this is standard practice on the LAN environments that are typical of VM deployments in data centers (SANs, Parallax [17], distributed file systems like Lustre), it is unacceptable for the high-latency WAN environments in which we envision Snowbird being used. A distributed storage mechanism is thus needed to take advantage of read and update locality in disk references. Furthermore,

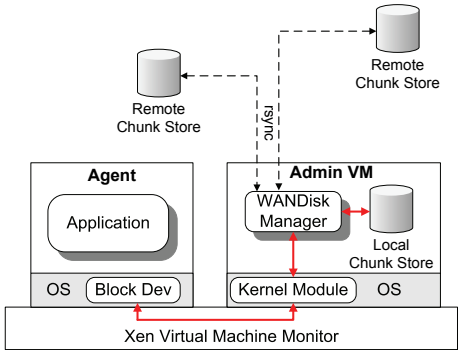


Fig. 5: WANDisk Storage System Architecture

while several WAN-optimized distributed storage choices were available to us, none of them satisfied two key characteristics of the Snowbird deployment model. First, there are multiple symmetric hosts on which an agent might run, thus precluding storage systems that are limited to a single replica support (DRBD), and systems that centralize data transfers on a server (NFS, AFS, Coda [18], VM disk mechanisms used by the Collective [7] and Internet Suspend/Resume [6], etc). Second, in this P2P-like model there is no need to maintain complex multiple-writer synchronization protocols [19], as the agent executes – and modifies its underlying disk state – in a single host at a time.

We have therefore implemented a distributed storage system called WANDisk, that provides efficient WAN access to multiple replicas of an agent's virtual disk. To provide flexibility in the choice of migration site, WANDisk follows a P2P approach where any Internet host can maintain a persistent replica of the agent's state. To reduce data transfers, WANDisk relies on the persistence of the replicas, which are created on demand as new migration sites are identified. WANDisk's replica control mechanism uses two techniques for optimizing the efficiency of agent migration. First, lazy synchronization is used to avoid unnecessary data transfers to inactive migration sites or for unused parts of a virtual disk. Second, differential transfers are used between replicas to reduce synchronization overhead.

Figure 5 shows the two-tiered WANDisk architecture, which consists of a *kernel module* and a user-space *disk manager*, both operating within Xen's administrative VM. The kernel module presents a pseudo block device that is mapped to an agent's virtual block device. All agent-originated block requests are handled by the pseudo block device and redirected into the user-space disk manager.

The disk manager partitions the agent's virtual disk into *chunks* and uses a *chunk table* to keep track of versioning and ownership information. Chunk size is configurable at agent creation time; we use a chunk size of 128 KB in our experiments, which we have found to work well in practice. As the agent modifies blocks in its virtual block device, the mapped chunk's version number is incremented, and its ownership transferred to the host where the agent is executing. Each host thus "owns" the chunks which the agent modified while executing there. Before the agent accesses any of those chunks at a different host, the chunk table will point WANDisk to the location of the freshest copy. The chunk table is thus the only piece of metadata necessary for the correct execution of WANDisk, and becomes a crucial addition to an agent's migratable state. To account for this, we have modified live migration in Xen to include

the chunk table; however, actual chunk transfers are not involved in the critical path of agent migration. WANDisk fetches chunks exclusively on-demand, using the rsync algorithm [20] to perform efficient differential data transfer.

The heavyweight `sync` command shown in Figure 1(b) is available for bringing any replica up to date under explicit user control. This command may be used for performance or reliability reasons. The command blocks until the replica at the specified migration site is both complete and up to date. At this point, agent execution can continue at that site even if it is disconnected from other replicas.

3 Usage Experience and Experimental Evaluation

We have gained hands-on usage experience with Snowbird by applying it to four bimodal applications from distinct application domains. None of these applications was written by us, and none had to be modified for use with Snowbird. Two of the applications (*Maya* and *ADF*) are commercial closed-source products whose success in the marketplace confirms their importance. The other two applications (*QuakeViz* and *Kmenc15*) have open source user communities. Section 3.1 describes these applications in more detail.

We found that using these applications with Snowbird was straightforward. Installing each as an agent was no more complex or time-consuming than installing it on a native machine. The only extra step was the creation of an application profile for the migration manager. Our generic application profile proved to be adequate for these four applications, but we recognize that some customization effort may be needed in the case of other applications.

This positive qualitative experience leads to a number of quantitative questions. What performance overheads does Snowbird incur? How much does it improve task completion time in the crunch phase, and crispness of interaction in the cognitive phase? How close is Snowbird's performance to that achievable through optimal partitioning (which is necessarily application-specific)?

The rest of this section describes our answers to these and related questions. Section 3.1 begins by describing the four applications and the benchmarks based on them. Section 3.2 then describes our approach to balancing realism and good experimental control in the cognitive phase, even in the face of unpredictable user behavior. Sections 3.3 and 3.4 describe our experimental setup. Finally, Section 3.5 presents our results.

3.1 Application Benchmarks

To demonstrate Snowbird's broad applicability, we experimented with applications that are representative of the domains of professional 3D animation, amateur video production, and scientific computing, and include both open source as well as commercial closed source products. For each application, we designed a representative benchmark that consists of a crunch and a cognitive phase.

Maya *(Digital Animation, closed source)*
This is a commercial closed source high-end 3D graphics animation package used for character modeling, animation, digital effects, and production-quality rendering [21] .

It is an industry standard employed in several major motion pictures, such as "*Lord of the Rings*," and "*War of the Worlds*." Our benchmark encompasses the typical work involved in completing an animation project. During the 29-minute cognitive phase, a digital character is loaded, a number of intermediate positions are generated by tweaking the character's skeleton and joints, and the animation pattern is scripted. The user periodically visualizes a low-fidelity preview of the animation, which Maya generates using graphics hardware acceleration. The crunch phase consists of rendering a photo-realistic version of each frame in the animation. This is a highly parallelizable CPU-intensive process that does not use graphics hardware. Maya allows the crunch phase to be initiated on a remote compute server, thus providing a case of application-specific partitioning against which to compare Snowbird.

QuakeViz *(Earth Sciences, open source)*
This is an interactive earthquake simulation visualizer, and the only benchmark that accesses a remote dataset. Our benchmark consists of the visualization of a 1.9 GB volumetric dataset depicting 12 seconds of ground motion around a seismic source in the Los Angeles Basin [22]. In our experiments, this dataset is stored on the remote compute server and accessed via NFS. During the crunch phase, QuakeViz mines the dataset to extract ground motion isosurfaces, surfaces inside the volume for which all points are moving in the same direction and at the same speed. The result is a set of triangular meshes depicting isosurfaces at successive time steps. Transformations such as smoothing and normals calculation are applied to the meshes to generate a more visually appealing result. In the cognitive phase, the isosurface meshes are rendered on the screen, and the user studies the seismic reaction by moving forwards or backwards in time and zooming, rotating, or panning the isosurfaces. Our benchmark explores 30 different time-steps during its 23-minute long cognitive phase.

ADF *(Quantum Chemistry, closed source)*
This is a commercial closed-source tool, used by scientists and engineers to model and explore properties of molecular structures [23]. In the ADF benchmark, the crunch phase consists of performing a geometry optimization of the threonine amino-acid molecule, using the Self-Consistent Field (SCF) calculation method. ADF distributes this intensive calculation to multiple CPUs using the PVM library, providing a second case of application-specific partitioning against which to compare Snowbird. The SCF calculation generates results that are visualized in a subsequent cognitive phase, such as isosurfaces for the Coulomb potential, occupied electron orbitals, and cut-planes of kinetic energy density and other properties. Analysis of these properties through rotation, zooming, or panning, are examples of the actions performed during the 26 minute-long cognitive phase.

Kmenc15 *(Video Editing, open source)*
This is an open-source digital editor for amateur video post production [24]. Users can cut and paste portions of video and audio, and apply artistic effects such as blurring or fadeouts. Kmenc15 can process and produce videos in a variety of standard formats. This benchmark does not exploit graphics hardware acceleration. In the 15-minute cognitive phase of our benchmark, we load a 210 MB video of a group picnic and split it into four episodes. We then edit each episode by cropping and re-arranging

portions of the recording and adding filters and effects. The crunch phase converts the four edited episodes to the MPEG-4 format. Kmenc15 converts the four episodes in parallel, exploiting available multiprocessing power.

3.2 Interactive Session Replay

One of the challenges in evaluating interactive performance is the reliable replay of user sessions. To address this problem, we developed *VNC-Redux*, a tool based on the VNC protocol that records and replays interactive user sessions. During the session record phase, VNC-Redux generates a timestamped trace of all user keyboard and mouse input. In addition, before every mouse button click or release, VNC-Redux also records a snapshot of the screen area around the mouse pointer. During replay, the events in the trace are replayed at the appropriate times. To ensure consistent replay, before replaying mouse button events the screen state is compared against the previously captured screen snapshot: if sufficient discrepancies are detected, the session must be reinitialized and replay restarted. Screen synchronization succeeds because VNC, like most other thin client protocols, is non work-conserving and can skip intermediate frame updates on slow connections. This results in the client always reaching a stable and similar (albeit not always identical) state for a given input. Therefore, given an identical initial application state, the entire recorded interactive session can be reliably replayed.

Unfortunately, the simple screen synchronization algorithms used by other replay tools [25] do not work well in high-latency environments. These algorithms typically perform a strict per-pixel comparison with a threshold that specifies the maximum number of pixel mismatches allowed. Something as simple as a mouse button release being delayed by a few milliseconds due to network jitter can cause a 3D object's position to be offset by a small amount. This offset causes the algorithm to detect a large number of pixel mismatches, stalling replay.

To address this problem, we developed an algorithm based on Manhattan distances to estimate image "closeness". For two pixels in the RGB color space, the Manhattan distance is the sum of the absolute differences of the corresponding R, G, and B values. If a pixel's Manhattan distance from the original pixel captured during record is greater than a given distance threshold, it is classified as a pixel mismatch. If the total number of pixel mismatches are greater than a pixel difference threshold, the screenshots being compared are declared to be different. Our experiments confirm that this improved matching algorithm works well over high latency networks.

3.3 Experimental Configurations

We investigate four configurations for executing bimodal applications:

- *Local Execution:* The application executes exclusively in an unvirtualized environment on a typical desktop-class machine. During interactive phases, 3D graphics are rendered using locally available hardware acceleration. This represents the best scenario for cognitive phases, but the worst case for crunch phases.
- *Remote Execution:* The application executes exclusively in an unvirtualized environment on an SMP compute server located behind a WAN link and close to external datasets. This represents the best scenario for crunch phases. As the user

interacts with the application over a WAN link using a standard VNC thin client, 3D rendering on the remote server is software based, representing the worst case for the cognitive phases.
- *Partitioned:* The application executes in an unvirtualized environment on the desktop-class machine, but is able to ship intensive computation to the remote compute server in an application-specific manner. This execution mode combines the best of remote and local execution, but is fully dependent on application support and requires multiple installations of the application. Not all of our benchmarks provide this mode of execution.
- *Snowbird:* Snowbird is used to dynamically switch between local and remote execution modes, independently of application support or lack of it. Both the user's desktop and remote compute server run the Snowbird infrastructure: Xen VMM, WANDisk, the hardware-accelerated agent GUI, and the migration manager. All benchmarks are initiated in an agent running at the user's desktop, with the WANDisk state at all hosts initially synchronized. The single generic application profile is used for all of our experiments.

By running the complete benchmark in each of the Remote and Local modes, we obtain two sets of results. First, a measure of what is clearly undesirable: running the crunch phase on an underpowered configuration (Local), and interacting with an application executing behind a WAN link (Remote). By comparing against these results we quantify the benefits of Snowbird in terms of reduced completion time for the crunch phase and improved interactive performance for the cognitive phase.

Conversely, the execution of the crunch and cognitive phases on the Remote and Local configurations, respectively, represents the ideal application partitioning. This provides an upper bound on the performance of any manual partitioning, as each phase is executed in the most advantageous location and no cost for communication overhead or added computational complexity is included. We compare Snowbird's performance against these set of results to quantify its overhead.

Finally, we can compare Snowbird to manual application partitioning for those applications that provide that option. While we expect manual application partitioning to be very close to optimal performance for both application phases, we also expect Snowbird to provide similar crunch and interactive performance.

3.4 Experimental WAN Testbed

Our experimental testbed consists of a user desktop, which is a 3.6 GHz Intel Pentium IV equipped with an ATI Radeon X600 Graphics Processor Unit (GPU), and a compute server, which is a four-way SMP (two dual-threaded cores) 3.6 GHz Intel Xeon. The desktop and server communicate through a NetEm-emulated WAN link with a bandwidth of 100 Mbit/s and RTTs of 33, 66, and 100 ms. These RTTs are conservative underestimates of the values observed between US and Europe, as shown in Table 1. We use a paravirtualized 2.6.12 Linux kernel for the Snowbird experiments and Fedora's 2.6.12 Linux kernel for the non-Snowbird experiments. Both kernels are configured with 512 MB of RAM. Agent VMs are configured as 512 MB SMP hosts, allowing them to fully utilize the computing power of the compute server's multiple cores. Snowbird uses the WAN-optimized HPN-SSH [26] protocol for data transfers.

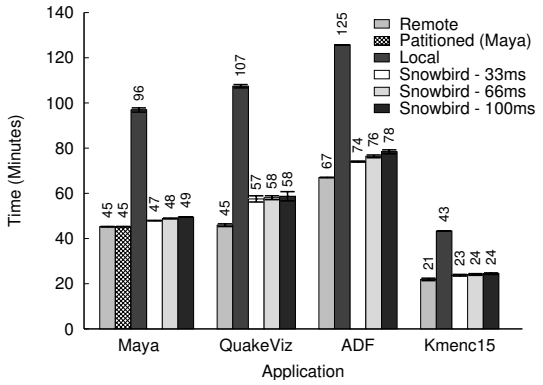


Fig. 6: Crunch Phase Completion Time

	Time (seconds)								
	Latency = 33 ms			Latency = 66 ms			Latency = 100 ms		
Application	Detect	Migrate	Suspend	Detect	Migrate	Suspend	Detect	Migrate	Suspend
Maya	10.8	51.9	3.5	10.8	53.5	4.7	11.5	58.2	5.6
QuakeViz	8.1	49.9	3.5	8.1	49.9	5.0	8.1	55.6	6.3
ADF	12.5	62.0	4.9	11.5	62.0	6.2	13.1	64.9	6.7
Kmenc15	8.1	51.8	4.7	9.1	54.0	5.7	8.4	59.5	6.7

Table 2: Crunch Phase Migration Time

3.5 Results

This section present the results of our experiments with the four benchmarks introduced in Section 3.1. All benchmarks include a cognitive and a crunch phase. In Maya and Kmenc15, the cognitive phase precedes the crunch phase, whereas in QuakeViz and ADF, the cognitive phase follows the crunch phase. Maya and ADF are the only applications we used that provide a partitioned execution mode. Unfortunately, the partitioned execution mode of ADF badly underperformed in our WAN testbed: with a 33 ms RTT, crunch phase completion time expands to roughly six times as much as in thin client mode. The vendor-supplied partitioning is designed for tightly-connected cluster computing and hence uses a very "chatty" synchronization protocol. This is an example of Snowbird overcoming the negative effects of an application-specific partitioning scheme that was not designed for WANs.

Crunch Phase Figure 6 shows the total completion time of the crunch phase for the benchmarks and configurations investigated. Each result is the mean of five trials; error bars show the observed standard deviations. For reasons explained earlier, partitioned execution results are not presented for ADF. As Figure 6 shows, Snowbird outperforms local execution by a significant margin. Since the impact of RTT on crunch phase performance is very small, we only show it for Snowbird. The crunch phases of all the benchmarks are CPU intensive and benefit from the increased computational power of the multiprocessor server. QuakeViz also takes advantage of the lower latency and increased bandwidth to its dataset, located on the compute server. More specifically, at 33 ms RTT, Snowbird approximately halves the length of the crunch phase for all applications, and comes within 4 to 28% of the ideal performance of the remote

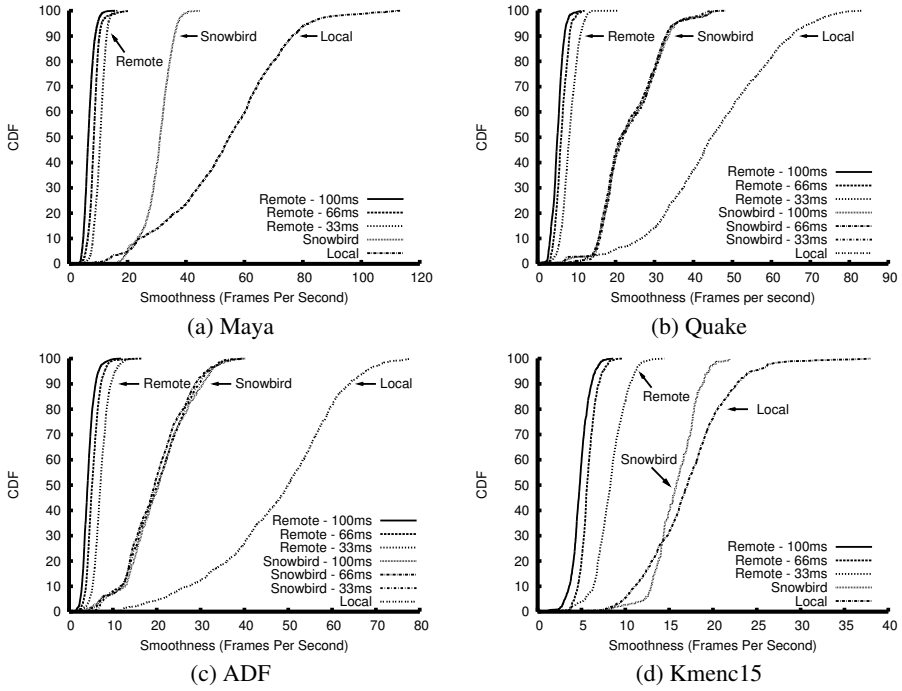


Fig. 7: Interactive Response

configuration. For Maya, it comes within 4 to 9% of the performance obtained through vendor-supplied partitioning.

Table 2 shows how long it takes the migration manager to detect the transition into the crunch phase, and how long it takes to migrate the agent to the remote compute server. Each result in this table is the mean of five trials, and the largest standard deviations observed for Detect, Migrate, and Suspend are 22%, 4%, and 7% of the corresponding means. As Table 2 shows, the maximum time taken by the migration manager is 14 seconds. Even with the worst-case latency of 100 ms, agent migration never takes more than 70 seconds to complete. In all cases, the agent spends less than 1.5 minutes on the user's desktop after it enters a crunch phase, which amounts to less than 5% of the total benchmark time. The table also shows that the maximum time for which an agent would appear to be unresponsive to user input during migration is six seconds or less. This is an order of magnitude smaller than the best value attainable without live migration (512 MB of VM RAM at 100 Mbit/s $\simeq$ 41 s).

Cognitive Phase Figure 7 shows the Cumulative Distribution Functions (CDFs) of the number of FPS per interaction for each of our four benchmarks under three configurations: local, remote, and Snowbird. Plots to the right indicate better performance than plots to the left. We show results for different network RTTs for the remote and Snowbird configurations. The cognitive phases for QuakeViz and ADF start on the remote compute server soon after the crunch phase terminates. The migration manager detects this transition and migrates back to the user's desktop. On the other hand, the cognitive phase of Maya and Kmenc15 start with the agent already running on the user's

desktop. We do not include results for Maya and ADF's partitioned mode, as they are practically identical to local interaction.

Our results show that Snowbird delivers a much better cognitive performance than remote interaction. More importantly, the median number of FPS delivered by Snowbird is above the long-established 20 FPS threshold needed for crisp interactivity [11]. In general, Snowbird's quantitative interactive performance is between 2.7 to 4.8 times better than that delivered by a thin client, with the interactive response in thin client mode rarely exceeding 10 FPS. Even though the agent has to migrate from the compute server to the user's desktop, Snowbird's cognitive performance tends to be independent of the WAN latency. Further, the network latency has a negligible impact on both the time taken before the decision to migrate is made and the time required to migrate the agent; we omit the migration time results for cognitive phases as they are very similar to those in Table 2.

The results also show that the FPS delivered by Snowbird is not as high as in unvirtualized local interaction. Local execution experiments delivered anywhere between 1.1 to 2.6 times more FPS in the median case. Nevertheless, once the agent migrates to the local host, in our subjective experience, the user experience delivered by Snowbird is indistinguishable from that of the native configuration for all of the benchmarks.

Summary Our results confirm that Snowbird offers significant benefits for bimodal applications. Without any application modifications, relinking, or binary rewriting, such applications are able to improve crunch performance through remote infrastructure. This improvement is achieved without compromising cognitive performance. Even when an application vendor supplies a partitioned mode of operation for cluster computing, Snowbird is able to offer comparable performance (within 4%), and in some cases greatly exceed its benefit over WANs.

4 Implementation Limitations and Future Extensions

The Snowbird prototype has certain limitations in functionality and performance. Some of these limitations arise from our goal of rapidly creating an experimental prototype rather than a robust, complete and efficient product. Other limitations have deeper roots in the design of Snowbird, and will therefore require more effort to overcome.

One limitation is that parallel Snowbird applications execute in a single SMP virtual machine. While the current trend of aggressively scaling processors to a hundred or more cores favors our design, some applications might be inherently designed to use multiple machines in a large cluster. Extending Snowbird to those applications would require new mechanisms such as "gang VM migration" that treat a group of VMs as a unit. We anticipate that these mechanisms will be conceptually simple, but their implementation may involve nontrivial complexity.

A second limitation arises from our use of hardware virtualization. At startup, an application might configure itself to take advantage of vendor-specific extensions to the x86 instructions set architecture, such as Intel's SSE or AMD's 3DNow!. Upon migration to different hardware, the application will crash when it attempts to execute an unsupported instruction. One possible solution is to use dynamic binary rewriting.

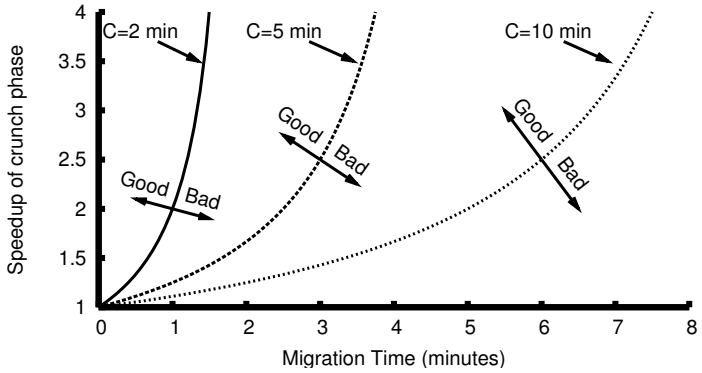


Fig. 8: When Is Snowbird Useful?

Another approach is to extend Snowbird so that it never attempts migration to an incompatible destination.

A third limitation is that Snowbird assumes a distinct separation of crunch and cognitive phases. Applications that consistently overlap these phases will not benefit from Snowbird. More generally, Snowbird is beneficial only when its agility of adaptation exceeds the rate of change of application behavior, and when remote execution provides sufficient improvement to overcome the cost of migration. Figure 8 illustrates this tradeoff for crunch phases. The horizontal axis shows migration time in minutes, which depends on the quality of the Snowbird implementation. This measure of system agility includes both the swiftness with which migration can be triggered, and the efficiency with which it can be completed. The vertical axis shows the crunch speedup when executing remotely, which depends on the application and the available remote resources. Each curve plots the relation $speedup = C/(C - migration\_time)$ for three hypothetical applications, where C is the crunch phase completion time when executing locally. Above each curve, Snowbird is beneficial; the expected performance gain exceeds the migration cost. Below each curve, Snowbird becomes harmful, as its migration cost eclipses any potential performance gains.

The simple model shown Figure 8 illustrates how improving migration time broadens the set of applications for which Snowbird is applicable. For a given speedup, workloads with smaller crunch time benefit as migration time decreases. And for a given crunch time, swifter migration reduces the constraints on the quality of the remote resources needed. Conversely, high migration times limit the applicability of Snowbird to applications with long crunch phases, or to remote platforms capable of yielding very high speedups. In the current prototype, detection and change of modality occur in roughly 10 seconds, while the migration that follows typically takes about 60 seconds plus lazy WANDisk chunk fetches. Mapping these values to Figure 8 indicates that crunch phases below ten minutes and speedups below a factor 2 will probably not show benefits with the current prototype.

It should be noted that a complementary attribute of agility is *stability,* which characterizes the ability of the implementation to avoid frivolous migrations that may

lead to thrashing. It is well known from control theory that agility and stability are two sides of the same coin, and have to be considered together in the design of an adaptive system. Improvements to Snowbird's agility may necessitate more sophisticated mechanisms for stability.

5 Related Work

To the best of our knowledge, Snowbird is the first system that exploits VM technology for the purpose of simplifying the development and deployment of bimodal applications. Closest in spirit to Snowbird is the large body of process migration research [9, 12, 27, 28]. Although extensively investigated for over two decades, no operating system in widespread use today supports process migration as a standard facility. We conjecture that this is because process migration is a brittle abstraction: a typical implementation involves so many external interfaces that it is easily rendered incompatible by a modest change. Snowbird implements a more resilient abstraction because the code and state implementing these interfaces is part of the OS that is transported with the application.

Language-based code mobility is another well-explored approach to moving computation. Relevant examples of work in this genre are Emerald [29] and one.world [30]. Java's remote method invocation framework has made this approach feasible and relevant to a range of computing environments. Snowbird's language-independent approach has the advantage of preserving substantial investments in legacy libraries, tool chains, and applications. It is also flexible with respect to code structure: an application can be a single monolithic process, or it can be a tool chain with scripts that glue the chain together. The crunch phase can have a finer structure, such as the use of multiple large datasets each of which is located at a different Internet site.

Snowbird can function as an adjunct to Grid computing middleware toolkits such as Globus [31] and Condor [32], that are widely used by the scientific computing community today. Snowbird complements the functionality provided by these toolkits by transforming a single monolithic application into an entity that can be easily migrated under toolkit control. More recently, the use of VMs has also been advocated for the Grid [14, 33], as enablers of simpler security and manageability abstractions.

Researchers have also developed toolkits for distributed visualization of large remote datasets. Examples include Dv [34], Visapult [2], SciRun [4], and Cactus [3]. Unlike Snowbird, these tools require applications to be written to a particular interface and are therefore useful only when application source code is available.

From a broader perspective, Snowbird was inspired by the substantial body of recent work on applying VM technology to a wide range of systems problems, including security [35], mobile computing [6, 36], and software maintenance [7]. Since the first technical report to describe our work [37], others have examined related techniques. Sandpiper [38] is a migration manager for cluster area networks that does not consider interaction-triggered relocations. Similar to VMGL, Blink [39] virtualizes GL-based 3D-rendering, but is specific to Xen-paravirtualized Linux. Bradford et al. [40] provide a virtual disk relocation scheme for the wide area with a full-prefetch policy that adds several minutes of downtime during migration.

6 Conclusion

A growing number of bimodal applications alternate between resource-intensive crunch phases and intensely interactive cognitive phases. The crunch phase may be CPU-intensive, memory-intensive, data-intensive, or some combination of all three. The cognitive phase must avoid sluggish or jerky responses in order to ensure low user distraction. This demands low end-to-end latency and may also require the use of local graphics acceleration hardware.

Snowbird simplifies the creation of bimodal applications by masking the distributed systems complexity of resource management, synchronization, and data consistency. It presents the simple programming abstraction of a VM to the developer, and assumes full responsibility for seamlessly migrating this VM to the best execution site. In experiments that include closed-source commercial applications, Snowbird offers crisp interactive performance that is superior to the best achievable through remote execution. At the same time, it is able to bring remote resources to bear on crunch phase performance. Without user or developer intervention, Snowbird is able to promptly detect application transitions between crunch and cognitive phases, and to automatically migrate the application to the most appropriate execution site.

An alternative viewpoint is to regard Snowbird as a tool that enables seamless transitions between the thin and thick client modes of execution. Thin clients are favored due to their ability to harness remote resources, while thick clients provide an unparalleled user experience during highly interactive tasks. By transparently morphing an application between the thick and thin client modes of execution, Snowbird gives a user the best of both worlds.

In closing, Snowbird's extensive use of multiple machines to meet the needs of a single user reflects an evolutionary trend that began with timesharing (fraction of a machine per user) and continued through personal computing (single machine per user) and client-server computing (local machine plus remote machines in fixed roles). Snowbird proposes a new step in this evolution by seamlessly and transparently using local and remote machines in flexible roles.

Acknowledgments

We thank Rajesh Balan for his involvement with earlier versions of this work. We also extend our gratitude to Nilton Bila, Angela Demke Brown, Debabrata Dash, Jan Harkes, Lionel Litty, Jing Su, Adin Scanell, and Alex Varshavsky for their feedback on early versions of this paper. We thank Beatriz Irigoyen for her help with ADF, Julio Lopez for his help with QuakeViz, Brian Paul for his help with Chromium, and Karan Singh for his help with Maya. Finally, we acknowledge the many and useful suggestions from the anonymous reviewers of this paper.

This research was supported by the National Science Foundation (NSF) under grant number CNS-0509004, the National Science and Engineering Research Council (NSERC) of Canada under grant number 261545-3 and a Canada Graduate Scholarship, by the Canadian Foundation for Innovation (CFI), and the Ontario Innovation Trust (OIT) under grant number 7739. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, NSERC, CFI, OIT, Carnegie Mellon University, or the University of Toronto. All unidentified trademarks mentioned in the paper are properties of their respective owners.

References

1. Afework, A., Beynon, M.D., Bustamante, F., Demarzo, A., Ferreira, R., Miller, R., Silberman, M., Saltz, J., Sussman, A., Tsang, H.: Digital Dynamic Telepathology - The Virtual Microscope. In: Proc. American Medical Informatics Association (AMIA) Annual Fall Symposium, Lake Buena Vista, FL (November 1998)
2. Bethel, W.: Visapult: A Prototype Remote and Distributed Visualization Application and Framework. In: Proc. SIGGRAPH Annual Conference, New Orleans, LA (July 2000)
3. Goodale, T., Allen, G., Lanfermann, G., Masso, J., Radke, T., Seidel, E., Shalf, J.: The Cactus Framework and Toolkit: Design and Applications. In: Proc. 5th International Conference on Vector and Parallel Processing - VECPAR, Porto, Portugal (June 2003)
4. Parker, S., Johnson, C.: SCIRun: A Scientific Programming Environment for Computational Steering. In: Proc. ACM/IEEE Supercomputing, San Diego, CA (December 1995)
5. National Laboratory for Applied Network Research (NLANR): RTT And Loss Measurements. `http://watt.nlanr.net/active/maps/ampmap_active.php`.
6. Satyanarayanan, M., Kozuch, M.A., Helfrich, C.J., O'Hallaron, D.R.: Towards Seamless Mobility on Pervasive Hardware. Pervasive and Mobile Computing **1**(2) (July 2005)
7. Sapuntzakis, C.P., Chandra, R., Pfaff, B., Chow, J., Lam, M.S., Rosenblum, M.: Optimizing the Migration of Virtual Computers. In: Proc. 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Boston, MA (December 2002)
8. Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live Migration of Virtual Machines. In: Proc. 2nd Symposium on Networked Systems Design and Implementation (NSDI), Boston, MA (May 2005)
9. Barak, A., Guday, S., Wheeler, R.G.: The MOSIX Distributed Operating System - Load Balancing for UNIX. Volume 672 of Lecture Notes in Computer Science. Springer-Verlag (June 1993)
10. Richardson, T., Stafford-Fraser, Q., Wood, K.R., Hopper, A.: Virtual Network Computing. IEEE Internet Computing **2**(1) (Jan/Feb 1998)
11. Airey, J.M., Rohlf, J.H., Frederick P. Brooks, J.: Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments. ACM SIGGRAPH Computer Graphics **24**(2) (March 1990)
12. Powell, M.L., Miller, B.P.: Process Migration in DEMOS/MP. In: Proc. 9th ACM Symposium on Operating Systems Principles (SOSP), Bretton Woods, NH (October 1983)
13. Lagar-Cavilla, H.A., Tolia, N., Satyanarayanan, M., de Lara, E.: VMM-Independent Graphics Acceleration. In: Proc. 3rd Conference on Virtual Execution Environments (VEE), San Diego, CA (June 2007)
14. Sundararaj, A.I., Dinda, P.A.: Towards Virtual Networks for Virtual Machine Grid Computing. In: Proc. 3rd Virtual Machine Research and Technology Symposium, San Jose, CA (May 2004)
15. Buck, I., Humphreys, G., Hanrahan, P.: Tracking Graphics State for Networked Rendering. In: Proc. ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware, Interlaken, Switzerland (August 2000)
16. VMware: `http://www.vmware.com/pdf/ws6_manual.pdf`.
17. Warfield, A., Ross, R., Fraser, K., Limpach, C., Hand, S.: Parallax: Managing Storage for a Million Machines. In: Proc. 10th Workshop on Hot Topics in Operating Systems (HotOS), Santa Fe, NM (June 2005)
18. Satyanarayanan, M.: The Evolution of Coda. ACM Transactions on Computer Systems **20**(2) (May 2002)
19. Muthitacharoen, A., Morris, R., Gil, T., Chen, B.: Ivy: A Read/Write Peer-to-peer File System. In: Proc. 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Boston, MA (December 2002)

20. Tridgell, A., Mackerras, P.: The rsync Algorithm. Technical Report TR-CS-96-05, Computer Science, Australian National University, Canberra, Australia (1996)
21. Maya: http://www.autodesk.com/maya.
22. Akcelik, V., Bielak, J., Biros, G., Epanomeritakis, I., Fernandez, A., Ghattas, O., Kim, E.J., Lopez, J., O'Hallaron, D., Tu, T., Urbanic, J.: High Resolution Forward and Inverse Earthquake Modeling on Terasacale Computers. In: Proc. ACM/IEEE Supercomputing, Phoenix, AZ (November 2003)
23. te Velde, G., Bickelhaupt, F.M., Baerends, E.J., Guerra, C.F., van Gisbergen, S.J.A., Snijders, J.G., Ziegler, T.: Chemistry with ADF. Computational Chemistry **22**(9) (July 2001)
24. Kmenc15: http://kmenc15.sourceforge.net/.
25. Zeldovich, N., Chandra, R.: Interactive Performance Measurement with VNCPlay. In: Proc. USENIX Annual Technical Conference, FREENIX Track, Anaheim, CA (April 2005)
26. Rapier, C., Stevens, M.: High Performance SSH/SCP - HPN-SSH, http://www.psc.edu/networking/projects/hpn-ssh/.
27. Douglis, F., Ousterhout, J.: Transparent Process Migration: Design Alternatives and the Sprite Implementation. Software Practice and Experience **21**(8) (August 1991)
28. Milojičić, D.S., Douglis, F., Paindaveine, Y., Wheeler, R., Zhou, S.: Process Migration. ACM Computing Surveys **32**(3) (September 2000)
29. Jul, E., Levy, H., Hutchinson, N., Black, A.: Fine-grained Mobility in the Emerald System. ACM Transactions on Computer Systems **6**(1) (February 1988)
30. Grimm, R., Davis, J., Lemar, E., Macbeth, A., Swanson, S., Anderson, T., Bershad, B., Borriello, G., Gribble, S., Wetherall, D.: System Support for Pervasive Applications. ACM Transactions on Computer Systems **22**(4) (November 2004)
31. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. Journal of Supercomputer Applications and High Performance Computing **11**(2) (June 1997)
32. Thain, D., Tannenbaum, T., Livny, M.: Distributed Computing in Practice: The Condor Experience. Concurrency and Computation: Practice and Experience **17** (Feb-Apr 2005)
33. Figueiredo, R.J., Dinda, P.A., Fortes, J.A.B.: A Case For Grid Computing On Virtual Machines. In: Proc. 23rd International Conference on Distributed Computing Systems (ICDCS), Providence, RI (May 2003)
34. López, J., O'Hallaron, D.: Evaluation of a Resource Selection Mechanism for Complex Network Services. In: Proc. IEEE International Symposium on High-Performance Distributed Computing (HPDC), San Francisco, CA (August 2001)
35. Dunlap, G.W., King, S.T., Cinar, S., Basrai, M.A., Chen, P.M.: ReVirt: Enabling Intrusion Analysis Through Virtual-machine Logging and Replay. In: Proc. 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Boston, MA (December 2002)
36. Kozuch, M., Satyanarayanan, M.: Internet Suspend/Resume. In: Proc. 4th IEEE Workshop on Mobile Computing Systems and Applications, Callicoon, NY (June 2002)
37. Lagar-Cavilla, H.A., Tolia, N., Balan, R., de Lara, E., Satyanarayanan, M., O'Hallaron, D.: Dimorphic Computing. Technical Report CMU-CS-06-123, Carnegie Mellon University (April 2006)
38. Wood, T., Shenoy, P., Venkataramani, A., Yousif, M.: Black-box and Gray-box Strategies for Virtual Machine Migration. In: Proc. 4th Symposium on Networked Systems Design and Implementation (NSDI), Cambridge, MA (April 2007)
39. Hansen, J.G.: Blink: Advanced Display Multiplexing for Virtualized Applications. In: Proc. 17th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), Urbana-Champaign, IL (June 2007)
40. Bradford, R., Kotsovinos, E., Feldmann, A., Schiöberg, H.: Live Wide-Area Migration of Virtual Machines including Local Persistent State. In: Proc. 3rd Conference on Virtual Execution Environments (VEE), San Diego, CA (June 2007)