Transparent Automatic Migration of Interactive Resource-Intensive Applications

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

January 2007
CMU-CS-07-101

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[*]Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA
[†]University of Toronto, Toronto, Canada

This research was supported by the National Science Foundation (NSF) under grant numbers CNS-0509004 and CCR-0205266, 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.

Keywords: Bimodal applications, Interactive, Resource-intensive, Crunch, Cognitive, Virtual Machine, VM Migration, Code Mobility, Application Partitioning

Abstract

Snowbird is a system that simplifies the development and use of applications that alternate between phases with heavy computational-resource needs and phases rich in user interaction. Examples of these applications include video editing and animation, as well as scientific, medical, and engineering diagnostic and design tools. Traditionally, these applications have been implemented as distributed programs. Snowbird, however, lets developers design their applications as monolithic units, and automatically migrates the application to the optimal execution site to achieve short completion time and crisp interactive performance. Snowbird augments VM migration with an automatic migration manager, graphics hardware acceleration, and a peer-to-peer storage system to accomplish these goals while avoiding the shortcomings that have limited the adoption of previous code mobility prototypes. Snowbird does not require that applications be written in a specific language, or use specific libraries, and can be used with existing applications. We present experimental results including some with closed-source commercial applications that validate Snowbird's approach to automatic migration.

1 Introduction

A growing number of applications alternate between resource-intensive *crunch phases* and intensely interactive *cognitive phases*. For example, character modeling using a 3D graphics animation package involves a cognitive phase in which the animator tweaks the character's skeleton to obtain desired intermediate positions and visualizes low-fidelity previews of the animation. This is followed by a crunch phase that generates a production-quality rendering of photo-realistic frames of the character's movements. Other examples include video editing in amateur and professional movie production, simulation and visualization of phenomena in scientific computing, computer assisted design in engineering and architecture, protein modeling for drug discovery in the pharmaceutical industry, and computer-aided diagnosis in medicine. We refer to this class of applications as *bimodal* applications.

Simultaneously optimizing performance for both phases of these applications is difficult. The crunch phase may be CPU-intensive, memory-intensive, data-intensive, or some combination of all three. This often leads to execution on a remote supercomputer. In some cases, the resource-intensive operations are performed on huge datasets that are too large to cache or mirror locally, or are constrained by organizational or regulatory policies that forbid the data copying implicit in caching and mirroring. The only option in that case is to execute the application at the dataset, perhaps located halfway across the world. Unfortunately, remote execution hurts interactive performance because of the harmful effects of Internet latency and jitter. Local execution is preferable in the cognitive phase for two reasons. First, it provides low-latency interaction that is unaffected by Internet load and congestion. Second, it enables use of local hardware for graphics performance acceleration.

These challenges complicate application development and slow the emergence of new applications. Today, developers address the very different demands of the crunch and cognitive phases by manually splitting the application into a distributed set of components [1, 5, 13, 31]. This approach requires developers to manage communication and coordination between the various application components, and forces them 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.

We have created a system called *Snowbird* that cleanly isolates intrinsic application complexity from the distractions of remote and local execution. Using Snowbird, a developer can focus on creating a bimodal application as a monolithic unit. At runtime, Snowbird automatically detects phase transitions and migrates the application to the optimal execution site. This achieves short completion time for crunch phases and crisp interactive performance during cognitive phases.

Snowbird exploits Virtual Machine (VM) technology to accomplish these goals. It wraps the application, including all its executables, scripts, libraries and configuration files into a migratable VM that we refer to as an *agent*. To support agents, we extended existing VM technology with three novel mechanisms: a migration manager to automatically trigger application relocations; support for the use of hardware-accelerated graphics by VM applications; and a peer-to-peer storage subsystem for the optimized transfer of persistent agent state.

To the best of our knowledge, Snowbird is the first system that exploits VM technology for the purpose of simplifying the development of bimodal applications. While Snowbird's functionality

is conceptually similar to process migration, the use of VM technology provides critical advantages in software deployment and maintenance. Although process migration has been extensively investigated for over two decades [4, 9, 28, 30, 33, 44, 48], no operating system in widespread use today (proprietary or open source) supports it 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 external change. Snowbird implements a more resilient abstraction because the code and state implementing these interfaces is part of the guest OS that is transported with the application. Snowbird also offers two additional advantages over previous approaches to mobile code. First, the application does not have to be written in a specific language (such as Java), nor does it need to be built using specific libraries. Second, existing closed-source applications can use Snowbird without recoding, recompilation, or relinking.

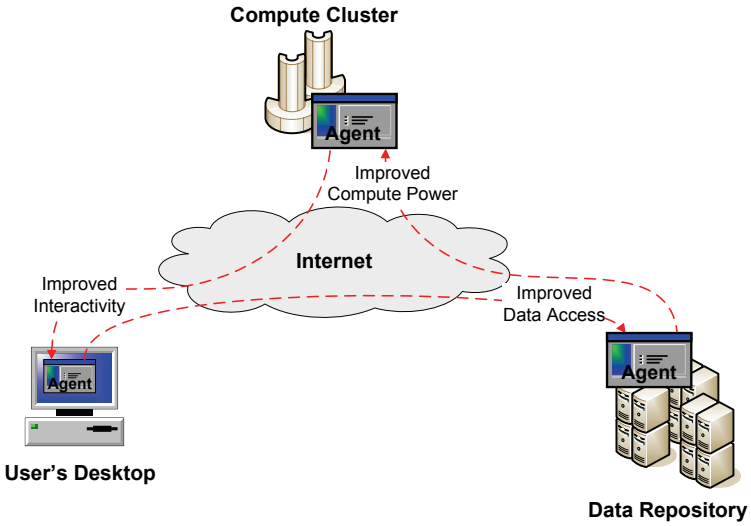
Experiments conducted with a number of real-world applications, including some that are commercial and closed-source, show that applications running under Snowbird typically come within 10% of optimal crunch completion times without compromising crisp interactive performance.

2 Background and Related Work

Closest in spirit to Snowbird is the large body of process migration research mentioned earlier. Unlike that body of work, Snowbird requires no host operating system changes. Rather than exposing numerous software interfaces that must be precisely matched, Snowbird builds on a very stable and rarely-changing interface to hardware.

Language-based code mobility is another well-explored approach to moving computation. The best early example of work in this genre is Emerald [20]. A more recent example is *one.world* [14]. The growth in popularity of Java and its support for *remote method invocation* [32] has made this approach feasible and relevant to a wide range of computing environments. Unfortunately, many established application domains are not Java-based; work in those domains involves legacy libraries and tool chains. In contrast, Snowbird does not require any modifications for legacy applications. Further, it does not require applications to be written in any particular language, and even the internal structure of the application is unconstrained. For example, the 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 a complement to *Grid computing toolkits* such as Globus [12] and Condor [43], that are widely used by the scientific computing community today. While there is considerable variation in the functionality provided by each toolkit, a representative sample includes finding idle machines, authenticating users, remotely executing an application on a machine, transferring results from one machine to another, checkpointing and restarting applications, and so on. A developer typically constructs a script or wrapper application that uses one of the above toolkits to chain together a sequence of individual computations and data transfers across a collection of machines. 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



This figure shows an example application that transitions through data- and compute-intensive phases before returning to the user for interaction-intensive usage.

Figure 1: Example of a Migrating Agent

recently, the use of VMs has also been advocated for the Grid [11, 21, 25, 41]. To simplify deployment, the complex middleware necessary for grid functionality can be easily packaged, configured and distributed as a VM.

Researchers have also developed toolkits for distributed visualization of large remote datasets. Examples include Dv [26], GVU [8], Visapult [5], SciRun [31], and Cactus [13]. Unlike Snowbird, these tools require their 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 [10], mobile computing [23, 39], and software maintenance [38]. Snowbird, however, is the first system to leverage VM functionality to simplify the development of bimodal applications.

3 Design and Implementation

Snowbird implements the notion of an *agent*, a migratable embodiment of an application that transparently and seamlessly relocates itself to achieve optimal performance. While an agent contains a single logical application, this application can be a tool chain made up of several processes executing simultaneously or sequentially in a pipeline fashion. Figure 1 shows the example of an agent that starts at the user's desktop, where execution of interactive cognitive phases is optimal. It then migrates to several remote sites to favor CPU performance or I/O performance, and then returns to the desktop for the next cognitive phase.

The figure also illustrates Snowbird's peer-to-peer (P2P) nature: agents execute on several hosts, each providing a desirable resource and separated by WAN links of potentially high latency.

Life Cycle Commands	Migration Commands	Administration Commands
`createagent` *agentname*	`suspend` *agentname*	`addhost` *agentname hostname*
`launch` *agentname*	`resume` *agentname hostname*	`sync` *agentname hostname*
`kill` *agentname*	`suspend-resume` *agentname hostname*	`movehome` *agentname newhome*
`purge` *agentname hostname*		`listhosts` *agentname*

Table 1: Snowbird Commands

Despite the symmetry of this setup, every agent has a unique *home* host, which acts as the authoritative machine on which commands used to modify agent state are issued. The home host is typically the user's local desktop or some other nearby computer where the user spends most of her time interacting with the agent. SSH access credentials are a necessary prerequisite for an agent to execute on other hosts; these SSH credentials are also used to encrypt all communications.

Table 1 shows the command line interface for Snowbird. It 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 3.2. However, they are available for explicit user or application control, if desired.

Snowbird offers four key advantages over existing approaches to code mobility. The rest of this section describes the implementation effort necessary to realize these advantages.

- First, applications do not have to be written in a specific language, nor do they need to be built using specific libraries.
- 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.
- Fourth, there is a clear separation between migration policy and mechanism. The code to decide when to trigger a migration is independent of the code that implements the migration.

3.1 Use of Virtual Machine Technology

Snowbird uses a Virtual Machine Monitor (VMM) to isolate each agent in its own VM. An agent can thus be any application binary, written in any programming language, running on any major OS. 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 [23, 38] 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* [7] 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.

In building Snowbird, our goal was to leverage existing VM technology as much as possible. In spite of this goal of minimalism, we found that the implementation effort was substantial. We had to add three new mechanisms to implement the agent abstraction: a migration manager that automatically triggers relocation; support for use of hardware-accelerated graphics by VM applications; and a peer-to-peer storage subsystem for persistent VM state. We discuss these in detail in the next three sections.

3.2 Sensor-Driven Migration Manager

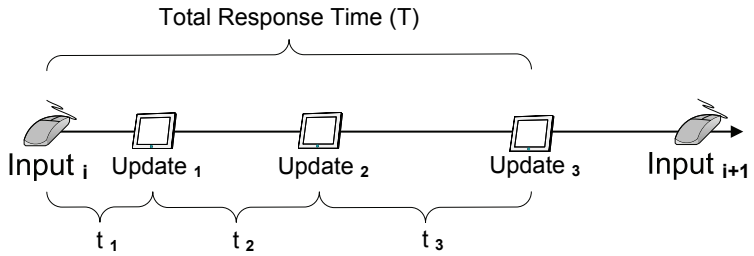
While applications can explicitly control migration decisions using the directives from Table 1, Snowbird provides system-controlled agent relocation as one of its key features. In other words, the decision to migrate, the choice of migration site, and the collection of information upon which to base these decisions can all happen under the covers in a manner that is transparent to the user and to the agent. Our solution uses a *migration manager* module. This module bases its migration decisions on *performance sensors* that extract relevant data from the VMM and the Snowbird user interface, 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. It also enables the use of migration managers based on entirely different principles.

3.2.1 Performance Sensors

Our implementation currently provides performance sensors for *CPU utilization*, *network utilization, interaction intensity,* and *interaction smoothness.* The CPU and network sensors periodically poll the VMM for CPU and network usage by a particular agent. The poll interval is configurable and has a default value of one second.

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 smoothness 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 *interaction smoothness* 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 the smoothness of an interactive response. Remote interaction usually relies on non-work-conserving thin-client algorithms such as VNC [36] that under adverse network conditions



This timeline shows the raw output of the interactivity sensor. Screen updates 1–3 are assumed to be causally related to the mouse input event Input$_i$. The resulting FPS is $3/T$.

Figure 2: Interaction Intensity and Smoothness

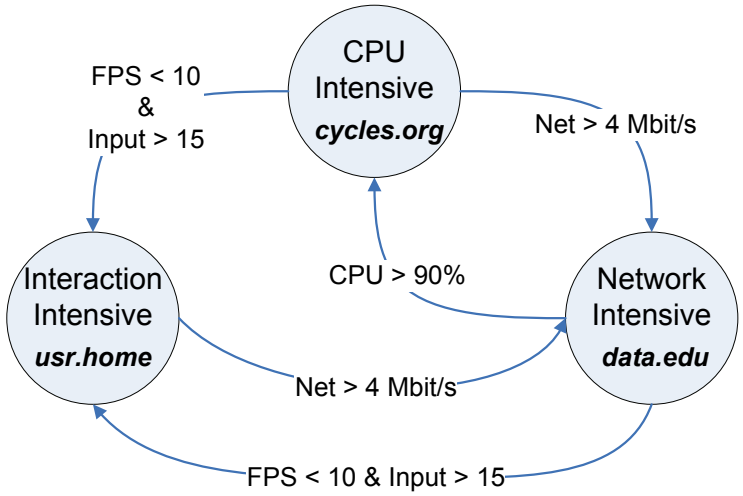
skip frames to "catch up" with the output. These low-FPS responses result in jerky on-screen tracking of mouse and keyboard inputs that can be annoying and distracting. We thus quantify the interaction smoothness of an event window as the average FPS yielded by all the inputs in that window. High interaction intensity combined with low interaction smoothness is the cue used by the migration manager to trigger a remote-to-local transition.

3.2.2 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. The state transition rules define when and how sensor readings should trigger state transitions. The profile also specifies the amount of past sensor information that should be used to evaluate the rules. Each state defines an optimal execution site. 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 or making decisions at a finer granularity.

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. Throughout the experiments in Section 5, we were able to use this generic application profile for all of our experiments. In less straightforward cases, machine learning techniques could be used to derive migration profiles.

A relevant concern 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 traditional hysteresis mechanisms [33], to shield the migration manager from adopting this erratic behavior.



Partial diagram of agent states and transitions. Each state includes its matching migration target (in boldface).

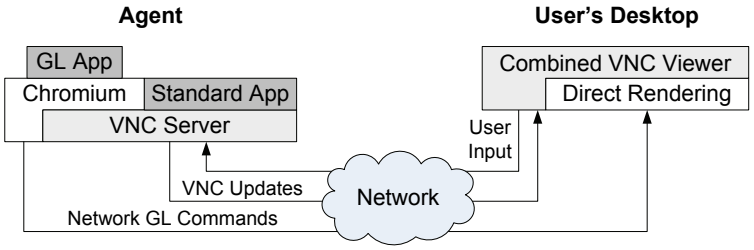
Figure 3: Example FSM of Agent States

3.3 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, whether the agent is running on the user's desktop or on a remote host. 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.

To meet these requirements Snowbird provides an enhanced thin client interface based on VNC [36]. When the agent is running on a remote host, the thin client protocol is used to communicate screen updates and user input events (i.e., keystrokes and mouse) over the network. When the agent is running on the user's desktop, the network becomes a loopback connection. Interaction is never interrupted as the agent is relocated because network connections persist through live-migrations: if the agent is relocated within the same subnet, a gratuitous ARP-reply binds the agent's IP address to the new physical host. Relocations across subnets are supported with VNETs [41], a Layer-2 proxy.

As efficient virtualization of graphics hardware requires manufacturer support, we instead virtualize the OpenGL API. This ubiquitous standard is supported by all major graphics hardware vendors and is the only pervasive cross-platform API for 3D applications. 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 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. Unlike other OpenGL API virtualization systems [15], ours does not require application or kernel modifications, and supports agent migration.



Using VNC as an example, this figure shows how 3D-intensive applications running within an agent can benefit from hardware acceleration found on a user's desktop.

Figure 4: Snowbird Extensions for 3D Support

The architecture of our OpenGL virtualization system is shown in Figure 4. We use Chromium [18], a framework that was originally designed for distributed rendering on clusters, as our OpenGL transport protocol. 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.

3.4 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 may be acceptable on the LAN environments that are typical of most VM deployments in data centers, it is unacceptable for the high-latency WAN environments in which we envision Snowbird being used. A distributed storage mechanism is needed to take advantage of read and update locality in disk references.

Distributed file systems are a mature technology today, with many systems in production use. Examples include NFS [37], AFS [17], Coda [40], and Lustre [6]. Storage Area Networks (SANs) have gained widespread popularity. Parallax [47] has been proposed as a storage subsystem for clusters of VMs. Unfortunately, most of these systems are designed for a LAN environment and perform poorly on WANs. Furthermore, client-server solutions that centralize all data transfers or support a single replica are inefficient given Snowbird's P2P usage model involving several hosts. These include file systems that perform acceptably on WANs, such as AFS and Coda, existing VM disk transfer mechanisms [39], and distributed block stores such as DRBD [35].

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.

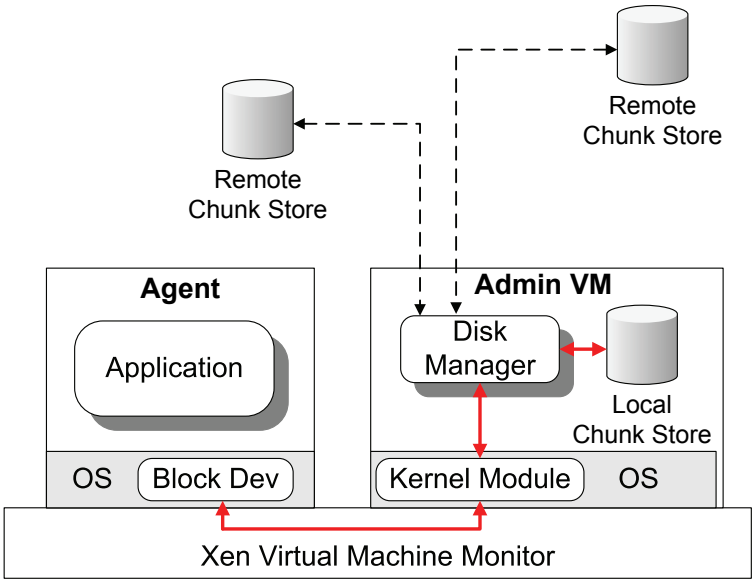


Figure 5: WANDisk Storage System Architecture

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 [45] to perform efficient differential data transfer.

The heavyweight `sync` command shown in Table 1 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.

3.5 Implementation Limitations

While the use of VM technology coupled with the additions described above enables seamless and transparent relocation of bimodal applications, Snowbird currently has some limitations that are the focus of our ongoing research.

The most relevant restriction is that Snowbird uses a shared memory model in which applications have to execute in a single SMP virtual machine. While Xen, our current choice of VMM, supports up to 32 processors, some highly parallel applications might require the use of multiple machines in a large cluster. As VMs running on cluster nodes are likely to be homogeneous, with the same kernel, loaded libraries, and running processes, we plan on exploiting this similarity among hosts to enable the concept of "gang migration," greatly amortizing the cost of migrating a large set of VMs.

The second restriction affects all systems that use VM migration techniques, and indeed all code migration prototypes that do not use interpreted languages. Vendor extensions to the x86 instructions set architecture, such as Intel's SSE and AMD's 3DNow!, can be problematic. At startup, an application might configure itself to take advantage of extensions available on one machine, and then crash upon migrating to another machine lacking those extensions. While dynamic binary rewriting could be employed, it could lead to a significant impact on performance. Instead, we propose extending the migration mechanism to check that all extensions available at the source are also available at the destination (i.e. only migrate to a superset). Given the ongoing efforts from vendors to support competing extensions, this is not an onerous restriction. For example, AMD has started supporting Intel's SSE3 extensions since the release of Athlon 64 chips in 2004.

Finally, an intrinsic limitation of our work is that it implicitly assumes a distinct separation of the crunch and cognitive phases. Applications that consistently overlap crunch and cognitive phases are not amenable to Snowbird.

4 Experimental Validation

Our experimental evaluation has two objectives. First, to determine the benefit of automatic migration compared to an static approach that runs both crunch and cognitive phases on the same location. This benefit is measured in terms of reduced completion time for the crunch phase and improved interactive performance for the cognitive phase. Second, to determine Snowbird's overhead compared to an ideal application partitioning that optimizes both the crunch and cognitive phases of the application.

In the rest of this section, we first describe our applications and the corresponding benchmarks we developed. This is followed by a description of the experimental testbed, and methodology. Finally, we describe VNC-Redux, a tool we implemented to enable consistent and repeatable replay of interactive user sessions.

Application	Domain	Source
Maya	Digital Animation	Closed
QuakeViz	Simulation Visualization	Open
ADF	Quantum Chemistry	Closed
Kmenc15	Video Editing	Open

Table 2: Application Characteristics

4.1 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. Table 2 summarizes the main characteristics of our applications.

4.1.1 Maya: Digital Animation

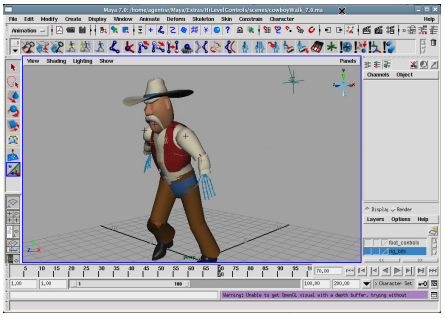
Maya [27] is a commercial closed source high-end 3D graphics animation package used for character modeling, animation, digital effects, and production-quality rendering. It is an industry standard and has been employed in several major motion pictures, such as "*Lord of the Rings*," "*War of the Worlds*," and "*The Chronicles of Narnia.*"

Our benchmark consists of a user loading a partially-complete animation project and completing it. The cognitive phase, which lasts for approximately 29 minutes, consists of specifying the degrees of freedom and motion bounds for the joints of a digital cowboy character (see Figure 6(a)), tweaking the character's skeleton to obtain desired intermediate positions, and scripting so that patterns of movement are rhythmically repeated. As part of this phase, the user periodically visualizes a low-fidelity preview of the animation. Maya leverages local graphics hardware for low-fidelity previews.

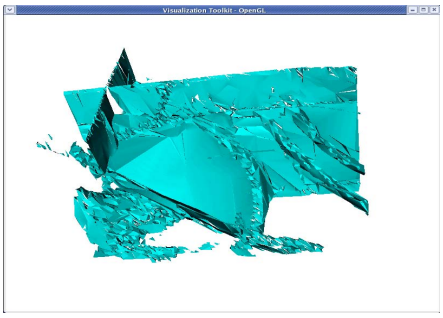
When the animation design is complete, the user initiates a production-quality rendering, i.e., the crunch phase. This is a parallelizable CPU-intensive task in which each photo-realistic frame is rendered with a number of lighting effects. Maya does not use any graphics hardware in this phase. The end-result is a collection of frames that can be encoded in any movie format.

4.1.2 QuakeViz: Simulation Visualization

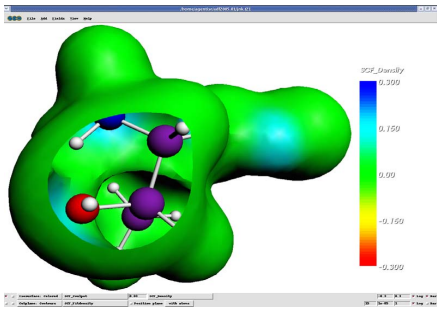
QuakeViz is an interactive earthquake simulation visualizer, and the only benchmark that accesses a remote dataset. Our benchmark consists of the visualization of a 1.9 GB volumetric dataset depicting 12 seconds of ground motion around a seismic source in the Los Angeles Basin [3]. During a computationally intense crunch phase, QuakeViz mines the dataset to extract ground motion isosurfaces. These are 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 representing an isosurface at successive points in time. A series of transformations including decimation, smoothing, and normals calculation, are then applied to generate a more visually appealing result.



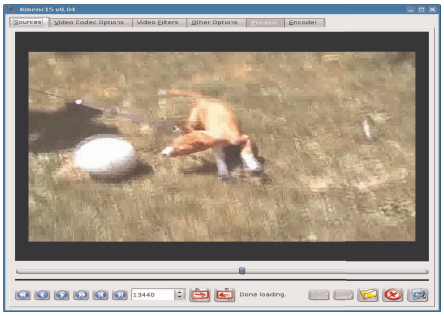
(a) Maya: Character modeling



(b) QuakeViz: Ground motion isosurface



(d) ADF: Energy density for an amino acid molecule



(c) Kmenc15: Video Editing

Figure 6: Application Screenshots

In the ensuing cognitive phase, the scene is synthesized and the meshes are rendered on the screen (see Figure 6(b)). During this phase, the user examines the rendered isosurfaces by zooming, rotating, panning, or moving forwards or backwards in time. The cognitive phase of the benchmark lasts for approximately 23 minutes, and involves exploration of the seismic reaction isosurfaces at 30 different time-steps.

4.1.3 ADF: Quantum Chemistry

Amsterdam Density Functional (ADF) [42] is a commercial closed-source tool, used by scientists and engineers to model and explore properties of molecular structures. 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.

The CPU intensive 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 (see Figure 6(c)). Analysis of these properties through rotation, zooming, or panning, are examples of the actions performed during the 26 minute-long cognitive phase.

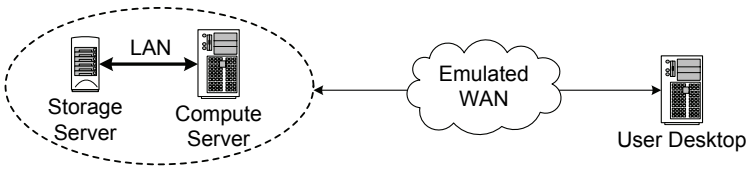


Figure 7: Experimental Testbed

4.1.4 Kmenc15: Video Editing

Kmenc15 (KDE Media Encoder) [22] is an open-source digital editor for amateur video post production. Users can cut and paste portions of video and audio, and apply artistic effects such as blurring or fadeouts. Kmenc15 can process AVI/MPEG-1 encoded video, and can export composite movies to a number of formats. This is the only benchmark that does not exploit graphics acceleration hardware.

In the cognitive phase of our benchmark, we load a 210 MB video of a group picnic and split it into four episodes (see Figure 6(d)). We then edit each episode by cropping and re-arranging portions of the recording and adding filters and effects. In all, the cognitive phase takes approximately 15 minutes. The user then starts the crunch phase by converting to MPEG-4 format all four edited episodes. As Kmenc15 can convert the four episodes in parallel, significant gains can be obtained from executing at a multiprocessor.

4.2 Experimental Testbed

Figure 7 shows our experimental testbed, which consists of a user desktop, a compute server, and a storage server. The user desktop is a 3.6 GHz Intel Pentium IV equipped with an ATI Radeon X600 Graphics Processor Unit (GPU). The compute server is a four-way SMP (two dual-threaded cores) 3.6 GHz Intel Xeon. The storage server is a PC that serves QuakeViz's dataset through a NFS share. Note that QuakeViz uses NFS only for its dataset: our policy was to serve external public datasets through well-known shared storage mechanisms, while using WANDisk for the virtual disk containing all the VM internal state, intermediate files generated by the applications, and final results.

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. Snowbird uses HPN-SSH [34], a WAN-optimized SSH variant, for all of its data transfers.

The user desktop communicates with the storage and compute servers through a WAN link emulated using NetEm [16]. Based on recent Internet bandwidth measurements on Planetlab [24] we configure the WAN link with a bandwidth of 100 Mbit/s. Table 3 shows NLANR [29] measurements of representative RTT latencies between Internet2 hosts in different cities in the US and Europe, leading to our choice of RTTs of 33, 66, and 100 ms. The storage and compute servers are connected via a Gigabit LAN.

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

These RTT measurements were obtained from NLANR [29]. The end hosts were all connected using high-bandwidth Internet2 links. The c column gives the lower bound RTT between the two endpoints at the speed of light.

Table 3: Internet2 Round Trip Times

4.3 Experimental Configurations

We investigate three configurations:

- **Local Execution:** The application executes exclusively in an unvirtualized environment on the user's desktop. During interactive phases, 3D graphics are rendered using locally available hardware acceleration. This represents the best scenario for the cognitive phase, but the worst case for the crunch phase.

- **Remote Execution:** The application executes exclusively in an unvirtualized environment on the SMP compute server. As all user interaction takes place over a standard VNC thin client, 3D rendering on the remote server is software based. This represents the best scenario for the crunch phase, but the worst case for the cognitive phase.

- **Snowbird:** Snowbird is used to dynamically switch between local and remote execution modes. 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. A 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, we quantify Snowbird's overhead by comparing it to the execution of the crunch and cognitive phases on the Remote and Local configurations, respectively. This second combination represents the ideal application partitioning, and provides an upper bound on the performance of any manual partitioning, as it does not include communication overhead or added computational complexity.

4.4 Interactive Session Replay

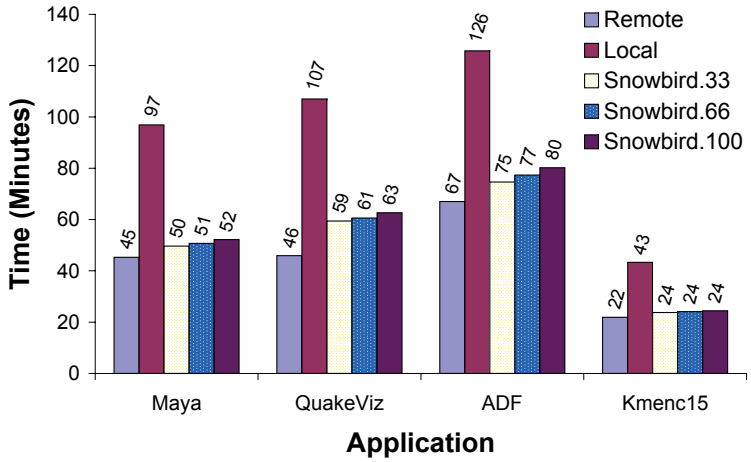
One of the challenges in evaluating interactive performance is the reliable replay of user sessions. To address this problem, we developed *VNC-Redux*, a tool based on the VNC protocol that records and replays interactive user sessions. During the session record phase, VNC-Redux generates a timestamped 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 [49] do not work well in high-latency environments. This algorithm performs 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 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.

5 Results

This section present the results of our experiments with the four benchmarks introduced in Section 4.1. 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.



This figure shows the crunch completion time for Local, Remote, and Snowbird experiments. Snowbird was evaluated at the 3 different WAN latencies. Results are the mean of three trials; maximum standard deviation was 2% of the corresponding mean.

Figure 8: Total Completion Time - Crunch Phase

	Time (seconds)									
	Best	Latency = 33 ms			Latency = 66 ms			Latency = 100 ms		
Application	Snowbird	Detect	Migrate	Suspend	Detect	Migrate	Suspend	Detect	Migrate	Suspend
Maya	2977	10.8	61.2	5.1	10.8	62.2	5.4	11.5	67.0	6.1
QuakeViz	3565	8.1	64.2	5.6	8.1	64.9	5.9	8.1	68.1	6.4
ADF	4478	12.5	63.3	5.5	11.5	61.0	5.4	13.1	65.2	6.8
Kmenc15	1424	8.1	51.8	4.7	9.1	54.0	5.7	8.4	59.5	6.7

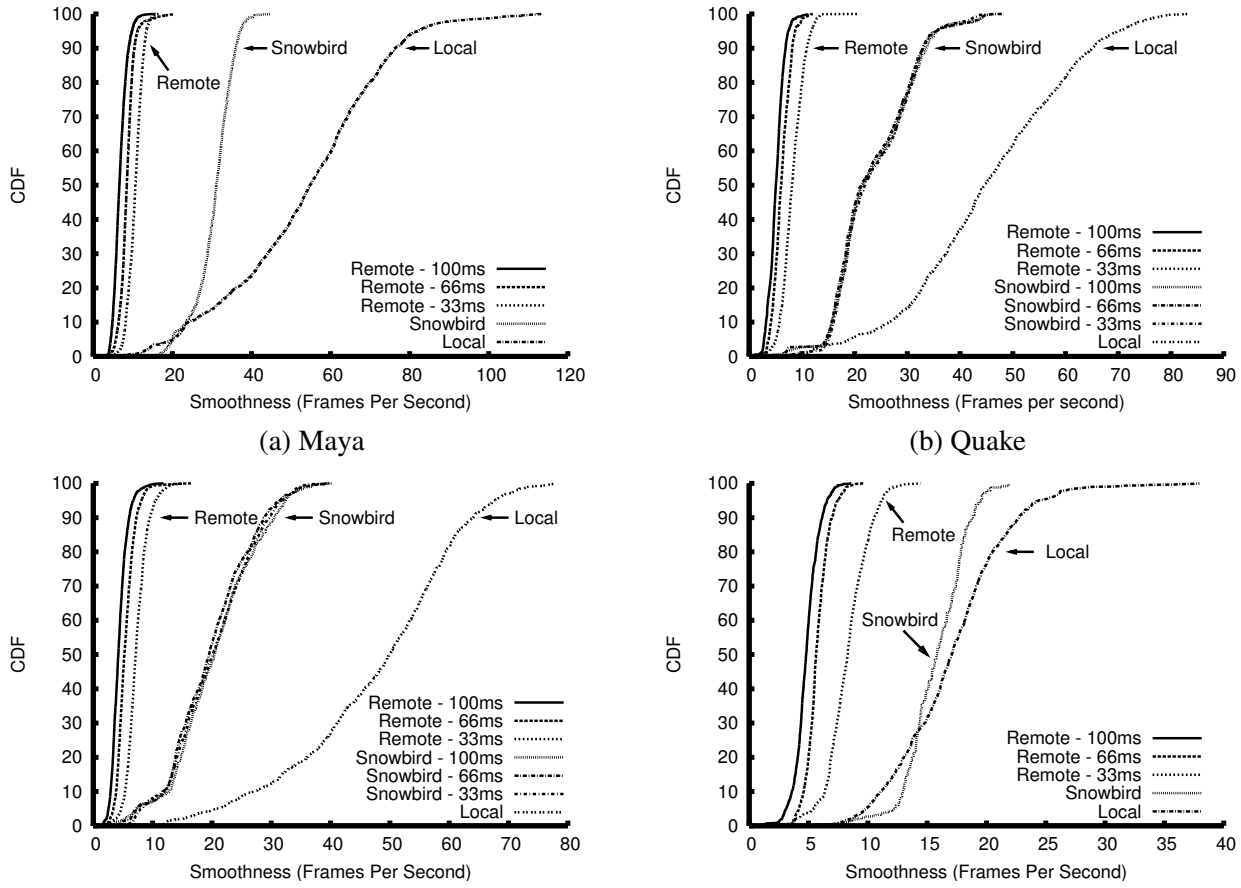
The *Detect*, *Migrate*, and *Suspend* columns measure the time taken by the migration manager to detect the transition to a crunch phase, time spent in live migration, and the time the agent was suspended for migration to complete. Results are the mean of three trials; maximum standard deviation for Detect, Migrate, and Suspend was 22%, 3%, and 11% of the corresponding means. For magnitude comparison, the *Best Snowbird* column shows the best observed crunch time for Snowbird.

Table 4: Crunch Phase Migration Times

5.1 Crunch Phase

Figure 8 shows the total completion time of the crunch phase for the four benchmarks under three configurations: local, remote and Snowbird. We only show results for different network round trip latencies for Snowbird, as the performance of the crunch phase for the local and remote configurations was unaffected by RTT. This was expected for Maya, ADF, and Kmenc15, which do not access external data. Since QuakeViz reads a large (1.9 GB) file over NFS, we expected that differences in network latency would affect its performance. However, it appears that readahead at the NFS client effectively masks the latency.

By migrating to the remote compute server, Snowbird is able to significantly outperform the local configuration. Specifically, at 33 ms, Snowbird approximately halved the length of the crunch phase for all applications, and came within 10 to 30% of the ideal performance of the remote configuration. 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



This figure shows the distribution of interactive responses for the cognitive phases of Maya, Quake, ADF, and Kmenc15. Snowbird results for Maya and Kmenc15 are independent of latency as they begin interaction in local execution mode and do not need to migrate.

Figure 9: Interactive Response

lower latency and increased bandwidth to the data server.

Table 4 shows the time it takes the migration manager to detect the transition into the crunch phase, and the time it takes to migrate the agent over to the remote compute server. The maximum time taken by the migration manager was 14 seconds. Further, even with the worst-case latency of 100 ms, agent migration never took more than 70 seconds to complete. In all cases, the agent spent less than 1.5 minutes on the user's desktop after it entered a crunch phase, 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 was seven seconds or less, an order of magnitude smaller than what an optimal non-live migration would entail ($512\mathrm{MB}/100\mathrm{Mbit/s} = 41\mathrm{seconds}$).

	Time (seconds)								
	Latency = 33 ms			Latency = 66 ms			Latency = 100 ms		
Application	Detect	Migrate	Suspend	Detect	Migrate	Suspend	Detect	Migrate	Suspend
Maya	*Not Relevant*								
QuakeViz	10.8	52.9	4.2	11.5	55.6	5.2	11.5	57.2	7.0
ADF	16.3	58.2	4.6	10.2	63.8	6.0	10.2	62.4	7.2
Kmenc15	*Not Relevant*								

The *Detect*, *Migrate*, and *Suspend* columns measure the time taken by the migration manager to detect the transition to a cognitive phase, time spent in live migration, and the time the agent was suspended for migration to complete. Maya and Kmenc15 results are not relevant as the workload does not include a cognitive phase after the crunch phase finishes at the remote server. Results are the mean of three trials; maximum standard deviation for Detect, Migrate and Suspend was 2%, 1%, and 12% of the corresponding means.

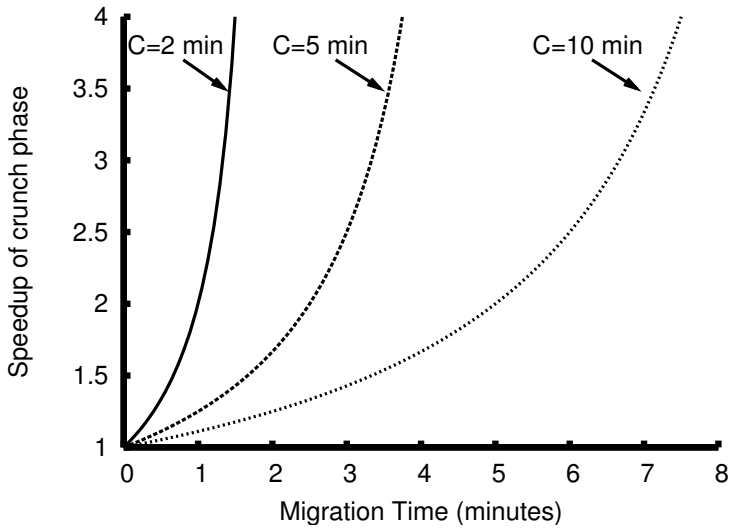
Table 5: Cognitive Phase Migration Times

5.2 Cognitive Phase

Figure 9 shows the Cumulative Distribution Functions (CDFs) of the number of frames per second (FPS) for each of our four benchmarks under three configurations: local, remote, and Snowbird. 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.

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 [2]. For example, results from the QuakeViz benchmark, seen in Figure 9 (b), show that in the median case, Snowbird delivers 2.7 to 4.3 times more FPS for the 33 and 100 ms latency cases respectively. In the 95^{th} percentile case, it delivers 3.0 to 4.8 times more FPS for the 33 and 100 ms latency cases respectively. 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. As Table 5 indicates, this occurs because 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.

However, the results also show the FPS delivered are not as high as those in unvirtualized local interaction. Local execution experiments delivered anywhere between 1.1 to 2.6 times more FPS in the median case and between 1.3 to 2.2 times more FPS in the 95^{th} percentile case. This difference between Snowbird and local execution is not a fundamental limitation of our design but is an artifact of the OpenGL virtualization described in Section 3.3. Chromium, the software used to intercept OpenGL calls, proved to be very CPU intensive. Adding another CPU core to the user's desktop or optimizing Chromium's implementation would bring Snowbird's performance much closer to that of unvirtualized local interaction.



This figure plots Snowbird's migration cost vs. application speedup, for the duration of the crunch phase (C) of three hypothetical applications. The parameter space *above* of each curve indicates where Snowbird is *beneficial*.

Figure 10: When Is Snowbird Useful?

6 Discussion and Improvements

The results presented in Section 5 demonstrate the feasibility and value of Snowbird for some real-world applications. It is important to note that none of these applications were written by us, or modified for use with Snowbird. Two of the applications (Maya and ADF) are commercial products whose success in the marketplace confirms their importance. The other two applications (QuakeViz and Kmenc15) have substantial open source user communities. All four are representative of a growing class of applications that embody distinct crunch and cognitive phases. To expand Snowbird's applicability to yet more applications, it is crucial to understand the tradeoffs inherent to its behavior.

Figure 10 illustrates the application space where Snowbird is applicable. 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. The rationale is that an agent executes locally at the user's desktop to provide the best possible interactive response, and when the application enters the crunch phase the agent migrates to a remote site if the expected performance gain exceeds the switching cost. Each curve in Figure 10 plots the relation $speedup < C/(C - migration\_time)$ for three hypothetical applications with crunch phases of different lengths C. This relation formalizes the behavior of Snowbird and splits the parameter space of the figure in two parts. Above each curve, Snowbird is beneficial; bellow the curve, it is harmful.

This simple model 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. A combination of short crunch time and modest crunch speedup defines a particularly challenging workload for Snowbird. 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 10 indicates that crunch phases below ten minutes and speedups below 2 are the approximate limits of applicability for Snowbird. We explore in the next section the most straightforward means to improve Snowbird's migration costs: attacking the WANDisk bottleneck.

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. Our current implementation has reasonable agility. While simple provisions have been adopted to deal with thrashing (see Section 3.2.2), we have not needed to use them throughout our experiments. We infer that the stability of our prototype is reasonable for the tested applications.

While harder to formalize, we can reasonably conclude that Snowbird's agility for cognitive phases is already beneficial for many applications. Even under challenging networking conditions in which a thin client would be painful for interactive use, a user can be confident that the agent will revert to local execution within roughly a minute. As mentioned in Section 5.2, there is still room for improvement when compared to an unvirtualized configuration. We plan to address this by streamlining the combined Chromium-VNC stack to significantly reduce processing overhead and latency.

6.1 Improving WANDisk Performance

Snowbird's performance is significantly affected by that of its storage subsystem, WANDisk. Our initial implementation, upon which the results of Section 5 are based, had many areas of inefficiency. First, the kernel module did not support concurrent requests. Second, there was significant overhead arising from the kernel module duplicating part of the functionality of Xen's virtual I/O subsystem. Third, a separate rsync process was forked for each chunk miss. Fourth, there was no prefetching or readahead within WANDisk.

We have recently implemented a new version, WANDisk2, that has a number of performance improvements. First, WANDisk2 uses asynchronous I/O to the local disk. Second, the kernel module now uses the blocktap [46] interface. Third, WANDisk2 opens persistent TCP connections to remote hosts, and fetches all chunks from a host using the same connection, without forking additional processes. Both rsync and raw byte copying can be used to transfer data. We have left prefetching for future work.

To quantify the effect of these improvements, we conducted a number of experiments using the synthetic IOZone benchmark [19]. We run four IOZone throughput benchmarks inside a Xen Linux VM for a 2 GB WANDisk block device: sequential read, sequential write, random read,

Test	Sequential		Random	
	Read	Write	Read	Write
No miss				
WANDisk1 (KB/s)	25194	8997	1864	1014
WANDisk2 (KB/s)	50063	43261	1977	8837
Speedup	**1.9**	**4.8**	**1.1**	**8.9**
5% miss				
WANDisk1 (KB/s)	1072	1123	673	433
WANDisk2 (KB/s)	10146	10501	1607	5936
Speedup	**9.5**	**9.3**	**2.4**	**13.7**
10% miss				
WANDisk1 (KB/s)	550	612	381	369
WANDisk2 (KB/s)	5447	5742	1437	4040
Speedup	**9.9**	**9.4**	**3.8**	**10.9**
20% miss				
WANDisk1 (KB/s)	291	307	241	273
WANDisk2 (KB/s)	2752	2897	1160	2401
Speedup	**9.5**	**9.4**	**4.8**	**8.8**
50% miss				
WANDisk1 (KB/s)	148	162	137	132
WANDisk2 (KB/s)	1097	1162	716	1047
Speedup	**7.4**	**7.2**	**5.4**	**7.9**
100% miss				
WANDisk1 (KB/s)	81	74	70	71
WANDisk2 (KB/s)	554	585	432	539
Speedup	**6.8**	**7.9**	**6.2**	**7.6**

Throughput comparison of WANDisk2 against the original WANDisk1. Speedup is the ratio of WANDisk2 performance to that of WANDisk1. Test size was 2GB and remote fetches happened through a 100 Mbit/s 66 ms RTT link. Each result is the mean of 3 trails. The maximum standard deviation for WANDisk1 and WANDisk2 was 5% and 3% of the corresponding mean, respectively.

Table 6: WANDisk Throughput Benchmark

and random write. While WANDisk uses the default 128 KB chunk size, IOZone performs I/O in units of 16 KBs. For each benchmark, we evaluate WANDisk with different hit ratios for its local chunk cache. On a chunk miss, WANDisk fetches the entire chunk from a remote cache through a 100 Mbit/s 66 ms RTT link.

Table 6 presents the results of the benchmark for both versions of WANDisk. WANDisk2 performance improvements are encouraging, as it consistently outperforms its predecessor by up to an order of magnitude. We expect that the good results of Section 5 will be even better with WANDisk2.

7 Conclusion

In this paper, we focus on a growing class of bimodal applications that alternate between resource-intensive (*crunch*) and interactive-intensive (*cognitive*) phases. We describe Snowbird, a system that simplifies the creation and use of bimodal applications by combining VM migration with support for VM 3D graphics hardware acceleration, a wide-area peer-to-peer storage system and an automatic migration manager. Snowbird lets the programmer focus on the application logic and develop a monolithic unit. During execution, Snowbird seamlessly and transparently relocates application execution. This allows easy and efficient use of remote resources such as compute servers and scientific datasets during crunch phases. It also provides crisp interactive performance during cognitive phases.

In experiments including closed-source commercial applications, Snowbird's interactive performance far exceeds that achievable through remote execution, while its crunch phase performance is significantly improved through the use of remote resources. Snowbird promptly detects application transitions between crunch and cognitive phases, and automatically migrates the application to the most appropriate execution site.

With the emergence of fields like computational biology, we expect the number, importance, and breadth of bimodal applications to grow substantially. Simultaneously, trends that promote the use of distant supercomputing centers are steadily increasing, along with the creation of large scientific datasets, all spread across the globe and connected by high-bandwidth high-latency links. It is in this challenging combination of circumstances that Snowbird's enabling role will find fertile ground.

Acknowledgments

We would like to thank Rajesh Balan for his involvement with earlier versions of this work. We would also like to thank Nilton Bila, Angela Demke Brown, Debabrata Dash, Jan Harkes, Jing Su, and Alex Varshavsky for their feedback on early versions of this paper. We would also like to 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.

References

[1] Asmara Afework, Michael D. Beynon, Fabian Bustamante, Angelo Demarzo, Renato Ferreira, Robert Miller, Mark Silberman, Joel Saltz, Alan Sussman, and Hubert Tsang. Digital dynamic telepathology - the virtual microscope. In *Proc. American Medical Informatics Association (AMIA) Annual Fall Symposium*, Lake Buena Vista, FL, November 1998.

[2] John M. Airey, John H. Rohlf, and Jr. Frederick P. Brooks. Towards image realism with interactive update rates in complex virtual building environments. In *SI3D '90: Proc. 1990 Symposium on Interactive 3D Graphics*, pages 41–50, Snowbird, UT, 1990.

[3] Volkan Akcelik, Jacobo Bielak, George Biros, Ioannis Epanomeritakis, Antonio Fernandez, Omar Ghattas, Eui Joong Kim, Julio Lopez, David O'Hallaron, Tiankai Tu, and John Urbanic. High resolution forward and inverse earthquake modeling on terasacale computers. In *Proc. ACM/IEEE conference on Supercomputing*, Phoenix, AZ, November 2003.

[4] Artsy, Y., Finkel, R. Designing a Process Migration Facility: The Charlotte Experience. *IEEE Computer*, 22(9):47–56, 1989.

[5] W. Bethel. Visapult: A prototype remote and distributed visualization application and framework. In *Proc. SIGGRAPH Annual Conference*, New Orleans, LA, July 2000.

[6] Peter J. Braam. The lustre storage architecture, November 2002. `http://www.lustre.org/docs/lustre.pdf`.

[7] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proc. 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, May 2005.

[8] K. Czajkowski, M. Thiebaux, and C. Kesselman. Practical resource management for grid-based visual exploration. In *Proc. IEEE International Symposium on High-Performance Distributed Computing (HPDC)*, San Francisco, August 2001.

[9] F. Douglis and J.K. Ousterhout. Transparent Process Migration: Design Alternatives and the Sprite Implementation. *Software Practice and Experience*, 21(8):1–27, 1991.

[10] George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza A. Basrai, and Peter M. Chen. Revirt: enabling intrusion analysis through virtual-machine logging and replay. In *Proc. 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December 2002.

[11] Renato J. Figueiredo, Peter A. Dinda, and José A. B. Fortes. A case for grid computing on virtual machines. In *Proc. 23rd International Conference on Distributed Computing Systems (ICDCS '03)*, page 550, Providence, RI, 2003.

[12] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2): 115–128, Summer 1997.

[13] T. Goodale, G. Allen, G. Lanfermann, J. Masso, T. Radke, E. Seidel, and J. Shalf. The cactus framework and toolkit: Design and applications. In *Vector and Parallel Processing - VECPAR '2002, 5th International Conference*. Springer, 2003.

[14] 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):421–486, 2004.

[15] Jacob Gorm Hansen. Blink: 3d multiplexing for virtualized applications. Technical Report 06-06, Dept. of Computer Science, University of Copenhagen, April 2006.

[16] Stephen Hemminger. Netem - emulating real networks in the lab. In *Proc. Linux Conference Australia*, Canberra, Australia, April 2005.

[17] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1), February 1988.

[18] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. In *Proc. 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 693–702, New York, NY, 2002.

[19] IOZone Filesystem Benchmark. IOZone. `http://www.iozone.org/`.

[20] Eric Jul, Henry Levy, Norman Hutchinson, and Andrew Black. Fine-grained mobility in the emerald system. *ACM Transactions on Computer Systems*, 6(1):109–133, 1988.

[21] Katarzyna Keahey, Ian Foster, Timothy Freeman, Xuehai Zhang, and Daniel Galron. Virtual workspaces in the grid. *Lecture Notes in Computer Science*, 3648:421–431, August 2005.

[22] Kmenc15. `http://kmenc15.sourceforge.net/`.

[23] Michael Kozuch and Mahadev Satyanarayanan. Internet suspend/resume. In *Proc. Fourth IEEE Workshop on Mobile Computing Systems and Applications*, Callicoon, New York, June 2002.

[24] Sung-Ju Lee, Puneet Sharma, Sujata Banerjee, Sujoy Basu, and Rodrigo Fonseca. Measuring bandwidth between planetlab host. In *Proc. 6th Passive and Active Measurement Workshop (PAM)*, Boston, MA, March 2005.

[25] Bin Lin and Peter Dinda. Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling. In *Proc. ACM/IEEE Conference on High Performance Networking and Computing (SC 2005)*, Seattle, WA, November 2005.

[26] Julio López and David O'Hallaron. 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.

[27] Maya. Maya. `http://www.autodesk.com/maya`.

[28] Dejan S. Milojičić, Fred Douglis, Yves Paindaveine, Richard Wheeler, and Songnian Zhou. Process migration. *ACM Computing Surveys*, 32(3):241–299, 2000. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/367701.367728.

[29] *RTT And Loss Measurements*. National Laboratory for Applied Network Research (NLANR), August 2006. `http://watt.nlanr.net/active/maps/ampmap_active.php`.

[30] Osman, S., Subhravati, D., Su., G., Nieh, J. The Design and Implementation of Zap: A System for Migrating Computing Environments. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, Boston, MA, December 2002.

[31] S. Parker and C. Johnson. Scirun: A scientific programming environment for computational steering. In *Proc. ACM/IEEE conference on Supercomputing*, San Diego, CA, December 1995.

[32] Pitt, E., McNiff, K. *java.rmi: The Remote Method Invocation Guide*. Addison-Wesley Professional, 2001.

[33] Michael L. Powell and Barton P. Miller. Process migration in demos/mp. In *Proc. 9th ACM Symposium on Operating Systems Principles (SOSP)*, October 1983.

[34] Chris Rapier and Michael Stevens. High Performance SSH/SCP - HPN-SSH, `http://www.psc.edu/networking/projects/hpn-ssh/`.

[35] Philipp Reisner. DRBD - Distributed replicated block device. In *Proc. 9th International Linux System Technology Conference*, Cologne, Germany, September 2002.

[36] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, Jan/Feb 1998.

[37] Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., Lyon, B. Design and Implementation of the Sun Network File System. In *Summer Usenix Conference*, Portland, OR, June 1985.

[38] Constantine P. Sapuntzakis, Ramesh Chandra, Ben Pfaff, Jim Chow, Monica S. Lam, and Mendel Rosenblum. Optimizing the migration of virtual computers. In *Proc. 5th Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[39] M. Satyanarayanan, Michael A. Kozuch, Casey J. Helfrich, and David R. O'Hallaron. Towards seamless mobility on pervasive hardware. *Pervasive and Mobile Computing*, 1(2): 157–189, 2005.

[40] Satyanarayanan, M. The evolution of coda. *ACM Transactions on Computer Systems*, 20(2), May 2002.

[41] Ananth I. Sundararaj and Peter A. Dinda. Towards virtual networks for virtual machine grid computing. In *Proc. 3rd Virtual Machine Research and Technology Symposium*, pages 177–190, San Jose, CA, May 2004.

[42] G. te Velde, F. Matthias Bickelhaupt, Evert Jan Baerends, Célia Fonseca Guerra, Stan J. A. van Gisbergen, Jaap G. Snijders, and T. Ziegler. Chemistry with ADF. *Journal of Computational Chemistry*, 22(9):931–967, 2001.

[43] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: The condor experience. *Concurrency and Computation: Practice and Experience*, 17:323–356, February-April 2005.

[44] Theimer, M., Lantz, K., Cheriton, D. Preemptable Remote Execution Facilities for the V-System. In *Proc. 10th Symposium on Operating System Principles (SOSP)*, Orcas Island, WA, December 1985.

[45] A. Tridgell and P. Mackerras. The rsync algorithm. Technical Report TR-CS-96-05, Department of Computer Science, The Australian National University, Canberra, Australia, 1996.

[46] Andrew Warfield, Steven Hand, Keir Fraser, and Tim Deegan. Facilitating the development of soft devices. In *Proc. USENIX Annual Technical Conference*, pages 379–382, Anaheim, CA, April 2005.

[47] Andrew Warfield, Russ Ross, Keir Fraser, Christian Limpach, and Steven Hand. Parallax: Managing storage for a million machines. In *Proc. 10th Workshop on Hot Topics in Operating Systems (HotOS)*, Santa Fe, NM, June 2005.

[48] Zandy, V.C., Miller, B.P., Livny, M. Process Hijacking. In *Proc. 8th International Symposium on High Performance Distributed Computing (HPDC)*, Redondo Beach, CA, August 1999.

[49] Nickolai Zeldovich and Ramesh Chandra. Interactive performance measurement with vncplay. In *Proc. USENIX Annual Technical Conference, FREENIX Track*, Anaheim, CA, April 2005.